第1章　PIC16F877 的外围功能模块

1.1.2　简单应用实例

该例用于令与 PORTD 口相连的 8 个发光二极管前 4 个点亮，后 4 个熄灭。在调试程序前，应使与 PORTD 口相连的 8 位拔码开关拔向相应的位置。

例 1.1　PORTD 输出

```
    #include  <pic.h>
main()
{
    TRISD=0X00；              /*TRISD 寄存器被赋值，PORTD 每一位都为输出*/
    while(1)；                /*循环执行点亮发光二极管的语句*/
    {
        PORTD=0XF0；          /*向 PORTD 送数据，点亮 LED（由实验模板*/
                              /*的设计决定相应位置低时 LED 点亮）。*/
    }
}
```

1.2.1　MSSP 模块 SPI 方式功能简介

下面是一段简单的 SPI 初始化例程，用于利用 SPI 工作方式输出数据的场合。

例 1.2　SPI 初始化程序

```
/*spi 初始化子程序*/
void    SPIINIT()
{
    PIR1=0；                      /*清除 SPI 中断标志*/
    SSPCON=0x30；             /* SSPEN=1；CKP=0 ，  FOSC/4 */
    SSPSTAT=0xC0；
    TRISC=0x00；              /*SDO 引脚为输出，SCK 引脚为输出*/
}
```

1.2.3　程序清单

下面给出已经在实验板上调试通过的一个程序，可作为用户编制其它程序的参考。

```
#include      <pic1687x.h>
/*该程序用于在 8 个 LED 上依次显示 1~8 等 8 个字符*/
static volatile int table[20]={0xc0，0xf9，0xa4，0xb0，0x99，0x92，0x82，0XD8，0x80，0x90，
0x88，0x83，0xc6，0xa1，0x86，0x8e，0x7f，0xbf，0x89，0xff}；
volatile    unsigned    char    data；
#define    PORTAIT(adr，bit)   ((unsigned)(&adr)*8+(bit))      /*绝对寻址位操作指令*/
static     bit PORTA_5 @ PORTAIT(PORTA，5)；
/*spi 初始化子程序*/
void    SPIINIT()
{
    PIR1=0；
    SSPCON=0x30；             /* SSPEN=1；CKP=0 ，  FOSC/4 */
    SSPSTAT=0xC0；
    TRISC=0x00；              /*SDO 引脚为输出，SCK 引脚为输出*/
}
/*系统各输入输出口初始化子程序*/
void    initial()
{
    TRISA=0x00；              /*A 口设置为输出*/
    INTCON=0x00；             /*关闭所有中断*/
    PORTA_5=0；                   /*LACK 送低电平，为锁存做准备*/
```

```
}
/*SPI 发送子程序*/
void            SPILED(int data)
{
      SSPBUF=data；                /*启动发送*/
      do
      {
            ；
      }while(SSPIF==0)；           /*等待发送完毕*/
      SSPIF=0；                    /*清除 SSPIF 标志*/
}
/*主程序*/
main()
{
      unsigned  I;
      initial()；                  /*系统初始化*/
      SPIINIT()     ；                   /*SPI 初始化*/
      for(i=8；i>0；i--)           /*连续发送 8 个数据*/
      {
            data=table[i]；              /*通过数组的转换获得待显示的段码*/
            SPILED(data)；         /*发送显示段码显示*/
      }
      PORTA_5=1；                        /*最后给锁存信号，代表显示任务完成*/
}
```

1.3.3　程序清单

下面给出已经在实验板上调试通过的程序，可作为用户编制其它程序的参考。有关显示部分的 SPI 初始化，请读者参考 1.2 节。

```
#include       <pic.h>
/*该程序用于按下相应的键时，在第一个 8 段 LED 上显示相应的 1～4 的字符*/
#define   PORTAIT(adr，bit)   ((unsigned)(&adr)*8+(bit))      /*绝对寻址位操作指令*/
static      bit   PORTA_5   @   PORTAIT(PORTA，5)；
#define   PORTBIT(adr，  bit)    ((unsigned)(&adr)*8+(bit))     /*绝对寻址位操作指令*/
static      bit   PORTB_5   @   PORTBIT(PORTB，5)；
static     bit   PORTB_4   @   PORTBIT(PORTB，4)；
static      bit   PORTB_1   @   PORTBIT(PORTB，1)   ；
static      bit   PORTB_2   @   PORTBIT(PORTB，2)   ；
unsigned       int   I；
unsigned       char j；
int   data；
 /*spi 初始化子程序*/
void       SPIINIT()
{
      PIR1=0；
      SSPCON=0x30；
      SSPSTAT=0xC0；
      TRISC=0xD7；                 /*SDO 引脚为输出，SCK 引脚为输出*/
}
/*系统各输入输出口初始化子程序*/
void       initial()
{
      TRISA=0xDF；
      TRISB=0XF0；                 /*设置与键盘有关的各口的数据方向*/
      INTCON=0x00；                /*关闭所有中断*/
```

```
	data=0X00；			/*待显示的寄存器赋初值*/
	PORTB=0X00；			/*RB1 RB2  先送低电平*/
	j=0；
}
/*软件延时子程序*/
void	DELAY()
{
	for(i = 6553；  --i  ；)
	continue；
}
/*键扫描子程序*/
int	KEYSCAN()
{
while(1)
	{
	if ((PORTB_5==0)||(PORTB_4==0))
	break；
}					/*等待有键按下*/
	DELAY()；			/*软件延时*/
	if ((PORTB_5==0)||(PORTB_4==0))
		KEYSERVE()；		/*如果仍有键按下，则调用键服务子程序*/
	else  j=0x00；			/*如果为干扰，则令返回值为 0*/
	return(j)；
}
/*键服务子程序*/
int	KEYSERVE()
{
	PORTB=0XFD	；
	if(PORTB_5==0)   j=0X01；
	if(PORTB_4==0)   j=0X03；
	PORTB=0XFB；
	if(PORTB_5==0)   j=0X02；
	if(PORTB_4==0)   j=0X04；/*以上根据按下的键确定相应的键值*/
	PORTB=0X00；			/*恢复 PORTB 的值*/
while(1)
	{
		if((PORTB_5==1)&&(PORTB_4==1)) break；/*等待键盘松开*/
	}
	return(j)；
}
/*SPI 发送子程序*/
void		SPILED(int data)
{
	SSPBUF=data；			/*启动发送*/
	do
	{
		；
	}while(SSPIF==0)；		/*等待发送完毕
	SSPIF=0；
}
/*主程序*/
main()
{
```

```
static  int table[20]={0xc0，0xf9，0xa4，0xb0，0x99，0x92，0x82，0XD8，0x80，0x90，
0x88，0x83，0xc6，0xa1，0x86，0x8e，0x7f，0xbf，0x89，0xff}；
    initial()；/*系统初始化*/
    SPIINIT()    ；/*SPI 初始化*/
while(1)
{
    KEYSCAN()；
    if(j!=0)                /*如果 j=0，证明先前的按键为干扰，则不予显示*/
    {
        data=table[j]；
        PORTA_5=0；           /*LACK 信号清 0，为锁存做准备*/
        SPILED(data)；
        PORTA_5=1；           /*最后给锁存信号，代表显示任务完成*/
    }
  }
}
```

1.4.1 PORTB 端口“电平变化中断”简介

例 1.3 PORTB 口“电平变化中断”初始化子程序

```
/*B 口“电平变化中断”初始化子程序*/
void        PORTBINT( )
{
    TRISB=0XF0；            /*设置相应口的输入输出方式*/
    OPTION=0x7F；           /*B 口弱上拉有效*/
    PORTB=0X00；            /*RB1，RB2 先送低电平*/
    RBIE=1；                /*B 口变位中断允许 */
    PORTB=PORTB；           /*读 B 口的值，以锁存旧值，为变位中断创造条件*/
}
```

1.4.3 程序清单

下面给出一个调试通过的例程，以供读者参考。有关显示的部分请读者参考前面章节。该程序中寄存器的位都用头文件中定义的位，如 RB5 表示 PORTB 的第 5 位，而不像前面几节那样自己定义。

```
    #include <pic.h>
/*该程序用于通过 PORTB 的"电平变化中断"进行键盘的识别。*/
/*程序设置一个键值寄存器 j，当按下 S9 键时 j=1，按下 S11 键时 */
/*j=2，按下 S10 键时，j=3，按下 S12 键时 j=4*/
unsigned    char    data；
unsigned    int     I；
unsigned    char    j；
const char table[20]={0xc0，0xf9，0xa4，0xb0，0x99，0x92，0x82，0XD8，0x80，0x90，0x88，
0x83，0xc6，0xa1，0x86，0x8e，0x7f，0xbf，0x89，0xff}；
/*B 口“电平变化中断”初始化子程序*/
void        PORTBINT()
{
    TRISB=0XF0；            /*设置相应口的输入输出方式*/
    OPTION=0x7F；
    PORTB=0X00；            /*RB1， RB2 先送低电平*/
    RBIE=1；                /*B 口变位中断允许 */
    PORTB=PORTB；           /*读 B 口的值，为变位中断创造条件*/
}
/*spi 初始化子程序*/
void        SPIINIT()
{
```

```
    PIR1=0；
    SSPCON=0x30；
    SSPSTAT=0xC0；
    TRISC=0xD7；                /*SDO 引脚为输出，SCK 引脚为输出*/
}
/*系统各输入输出口初始化子程序*/
void    initial()
{
    TRISA=0xDF；
    INTCON=0x00；               /*关闭所有中断*/
    data=0X00；                 /*待显示的寄存器赋初值*/
}
/*键服务子程序*/
void    KEYSERVE()
{
    PORTB=0XFD    ；
    if(RB5==0)   j=0X01；
    if(RB4==0)   j=0X03；
    PORTB=0XFB    ；
    if(RB5==0)   j=0X02；
    if(RB4==0)   j=0X04；           /*以上通过逐行逐列扫描，以确定是何键按下*/
    PORTB=0X00；                /*恢复 PORTB 的值*/
}
/*软件延时子程序*/
void        DELAY()
{
    for(i = 6553；  --i  ；)
    continue；
}
/*SPI 发送子程序*/
void    SPILED(int data)
{
    SSPBUF=data；               /*启动发送*/
    do
    {
        ；
    }while(SSPIF==0)；
    SSPIF=0；
}
void    IDEDIS()
{
    KEYSERVE()；                /*进行键盘的识别*/
    data=table[j]；                 /*获得需要送出显示的段码*/
    RA5=0；                     /*LACK 信号清 0，为锁存做准备*/
    SPILED(data)；
    RA5=1；                     /*最后给一个锁存信号，代表显示任务完成*/
}
/*中断服务程序*/
void interrupt  keyint(void)
{
    DELAY()；                   /*软件延时*/
    if ((RB5==0)||(RB4==0))         /*该语句除了能够确认按键是否为干扰外，*/
                                /*还可以屏蔽一次键松开时引起的中断*/
```

```
        IDEDIS()；                  /*键识别和显示模块*/
        PORTB=PORTB；               /*读 B 口的值，改变中断发生的条件，避免键*/
                                    /*一直按下时，连续进行键识别*/
        RBIF=0；                    /*键扫描时可能会产生"电平变化"而使 RBIF*/
                                    /*置 1，再清除一次 RBIF 以避免额外中断*/
}
main()
{
        initial()；                 /*系统初始化*/
        PORTBINT()；                /*B 口变位中断初始化*/
        SPIINIT()     ；                /*利用 SPI 显示初始化*/
        ei()；                      /*总中断允许*/
while(1)
        {
            ；
        }                           /*等待中断*/
}
```

1.5.2　程序清单

下面给出一个调试通过的例程，可作为读者的参考。调试该程序把模板 J7 上的短路跳针拔下，以免产生冲突。

```
#include  <pic1687x.h>
volatile  unsigned      char      data；
/*spi 初始化子程序*/
 void     SPIINIT()
{
        PIR1=0；
        SSPCON=0x30；               /* SSPEN=1；CKP=0 ，  FOSC/4 */
        SSPSTAT=0xC0；
        TRISC=0x10；                /*SDI 引脚为输入，SCK 引脚为输出*/
}
/*系统各输入输出口初始化子程序*/
void      initial()
{
        TRISA=0x00；
        TRISD=0x00；                /*D 口为输出方式*/
        INTCON=0x00；               /*关闭所有中断*/
}
/*SPI 接收子程序*/
 int      SPIIN()
{
        RA4=0；                     /*74HC165 并行置数使能，将 8 位开关量置入器件*/
                                    /* (LOAD 为低电平时 8 位并行数据置入 74HC165)*/
        RA4=1；                     /*74HC165 移位置数使能(LOAD 为高电平时芯*/
                                    /*片才能串行工作)*/
        SSPBUF=0；                  /*启动 SPI，此操作只用于清除 SSPSTAT 的
                                    *BF 位，因此 W 中的实际数据无关紧要*/
        do{
            ；
        }while(SSPIF==0)；          /*查询数据接收完毕否？*/
        SSPIF=0；
        data=SSPBUF；
        return(data)；              /*返回接收到的数据*/
}
```

```
/*把 SPI 接收的数据通过 D 口显示在 8 个发光二极管上的子程序*/
void        SPIOUT(int data)
{
    PORTD=~data；
}
/*主程序*/
main( )
{
    initial()；                    /*系统初始化*/
    SPIINIT()；                    /*SPI 初始化*/
 while(1)
  {
    SPIIN()；                          /*SPI 接收外部数据*/
    SPIOUT(data)；                 /*送出数据显示*/
  }
}
```

1.2.1 CCP 模块的 PWM 工作方式简介

下面给出一个 CCP 模块设置为 PWM 操作时的初始化程序

例 1.4 CCP 模块设置为 PWM 方式时的初始化程序

```
/*CCP1 模块的 PWM 工作方式初始化子程序*/
void        CCP1INIT()
{
 CCPR1L=0X7F；
 CCP1CON=0X3C；                 /*设置 CCP1 模块为 PWM 工作方式，且其工作循环
                                *的低 2 位为 11，高 8 位为 01111111=7F*/
 INTCON=0X00；                      /*禁止总中断和外围中断*/
 PR2=0XFF；                         /*设置 PWM 的工作周期*/
 TRISC=0XFB；                   /*设置 CCP1 引脚为输出方式*/
}
```

该初始化子程序设置 CCP1 模块输出分辨率为 10 位的 PWM 波形，且占空比为 50%。

1.2.3 程序清单

下面给出一个调试通过的例程，可作为读者编制程序的参考。

```
#include  <pic.h>
/*该程序用于使 CCP1 模块产生分辨率为 10 位的 PWM 波形，占空比为 50%*/

/*CCP1 模块的 PWM 工作方式初始化子程序*/
void CCP1INIT()
{
 CCPR1L=0X7F；
 CCP1CON=0X3C；                 /*设置 CCP1 模块为 PWM 工作方式，且其工作
                                *循环的低 2 位为 11，高 8 位为 01111111=7F*/
 INTCON=0X00；                      /*禁止总中断和外围中断*/
 PR2=0XFF；                         /*设置 PWM 的工作周期*/
 TRISC=0XFB；                   /*设置 CCP1 引脚为输出方式*/
}
/*主程序*/
main()
{
    CCP1INIT()；                       /*CCP1 模块的 PWM 工作方式初始化*/
    T2CON=0X04；                   /*打开 TMR2，且使其前分频为 0，
                                       *同时开始输出 PWM 波形*/
do
    {
```

```
		;
	}while(1)；			/*系统开始输出 PWM 波形。如果系统是
					*多任务的，则可以在此执行其它任务，而
					*不会影响 PWM 波形的产生*/
}
```

1.3.3　应用程序

2.　程序清单

```
#include	<pic.h>
/*此程序实现"看门狗"WDT 的功能*/
unsigned	long	I；
/*系统初始化子程序*/
void		initial()
{
	OPTION = 0X0F；			/*把前分频器分配给 WDT，且分频倍率为 1:128*/
	TRISD = 0X00；			/*D 口设为输出*/
}
/*延时子程序*/
void	DELAY()
{
 for (i=19999；--i；)
	continue；
}
/*主程序*/
main ()
{
	initial()；			/*初始化，设定看门狗的相关寄存器*/
	PORTD = 0X00；			/*D 口送 00H，发光二极管亮*/
	DELAY()；			/*给予一定时间的延时*/
	PORTD = 0XFF；			/*D 口送 FFH，发光二极管灭*/
while(1)
	{
		;
	}				/*死循环，等待看门狗溢出复位*/
}
```

1.4.3　程序清单

该例在 PIC16F877 休眠前使 8 个发光二极管的高 4 个发光，然后进入休眠工作方式；若按键引起的中断将其激活，则低 4 个发光。用 C 语言编写程序时，语句 SLEEP（）相当于汇编语言中的语句“ sleep”，使单片机进入休眠状态。

```
#include	<pic.h>
/*该程序实现 PIC16F877 的休眠工作方式，并由实验板上的按键产生"电平变化中断"将其*
从休眠状态中激活。休眠与激活的状态由与 D 口相连的 8 个 LED 显示。休眠时高 4 个
*LED 发光，低 4 个 LED 熄灭；  激活以后高 4 个 LED 熄灭，低 4 个 LED 发光*/
unsigned  long	i；
/*系统初始化子程序*/
void initial()
{
		di()；			/*全局中断禁止，"电平变化中断"只执行唤醒功能*/
		RBIE=1；			/*PORTB 口电平变化中断允许*/
		RBIF=0；			/*清除 B 口电平变化中断标志*/
		TRISB4=1；
		TRISB5=1；
		TRISB2=0；
```

```
        TRISB1=0；           /*设置与键盘有关的各 I/O 口的输入输出方式*/
        TRISD=0X00；         /*D 口为输出*/
        PORTB=0X00；         /*键盘的行线送低电平，为“ 电平变化中断 ” 作准备*/
        PORTB=PORTB；        /*读 PORTB 的值，锁存旧值，也为“ 电平变化
                             *中断 ” 作准备*/
}
/*主程序*/
main ()
{
        initial()；          /*初始化*/
        PORTD=0X0F；            /*高 4 个 LED 灯亮*/
        SLEEP()；            /*单片机开始进入休眠状态*/
        PORTD=0XF0；            /*激活后，低 4 个 LED 灯亮*/
while(1)
        {
          ；
        }
}
```

NOTE：

第2章　模拟量输入与输出

2.1　A/D 转换的应用

例 2.1　A/D 转换初始化程序

```
//A/D 转换初始化子程序
void      adinitial( )
{
      ADCON0 = 0x51；          //选择 A/D 通道为 RA2，打开 A/D 转换器
                               //在工作状态，且使 AD 转换时钟为 8tosc
      ADCON1 = 0X80；          //转换结果右移，及 ADRESH 寄存器的高 6 位为"0"
                               //且把 RA2 口设置为模拟量输入方式
      PIE1 = 0X00；
      PIE2 = 0X00；
      ADIE = 1；               //A/D 转换中断允许
      PEIE = 1；               //外围中断允许
      TRISA2=1；               //设置 RA2 为输入方式
}
```

2.1.2　程序清单

下面给出一个调试通过的例程，可作为读者编制程序的参考。该程序中用共用体的方式把 A/D 转换的 10 位结果组合在一起。有关共用体的详细资料请参考本书相关章节。

```
# include      <pic.h>
union          adres
{int      y1；
unsigned   char   adre[2]；
}adresult；                    //定义一个共用体，用于存放 A/D 转换的结果
unsigned    char    i；
unsigned    int     j；
//系统各 I/O 口初始化子程序
void      initial()
{
      TRISD=0X00；             //D 口为输出
      i=0x00；
}
//A/D 转化初始化子程序
void       adinitial()
{
      ADCON0=0x51；            //选择 A/D 通道为 RA2，打开 A/D 转换器
                               //在工作状态，且使 A/D 转换时钟为 8tosc
      ADCON1=0X80；            //转换结果右移，及 ADRESH 寄存器的高 6 位为"0"
                               //且把 RA2 口设置为模拟量输入方式
      PIE1=0X00；
      PIE2=0X00；
      ADIE=1；                 //A/D 转换中断允许
      PEIE=1；                 //外围中断允许
      TRISA2=1；               //设置 RA2 为输入方式
}
//延时子程序
void      delay()
{
```

```
        for(j=5535；--j；) continue；
}
//报警子程序
void        alarm()
{
        i=i^0xFF；                    //通过异或方式每次把 i 的各位值取反
        PORTD=i；                     //D 口输出 i 的值
}
//中断服务程序
void        interrupt        adint(void)
{
        ADIF=0；                      //清除中断标志
        adresult.adre[0]=ADRESL；
        adresult.adre[1]=ADRESH；//读取并存储 A/D 转换结果，A/D 转换的结果通过共
                                      //用体的形式放入了变量 y1 中
        if(adresult.y1>0x200)
        {
        alarm()；                     //如果输入的模拟量大于 2.5V(对应数字量
                                      //0X200h)，则调用报警子程序
        delay()；                     //调用延时子程序，使电压检测不要过于频繁
        }
        else  PORTD=0XF0        ；        //如果输入的模拟量小于 2.5V，则与 D 口相连的
                                      //8 个发光二极管的低 4 个发亮，表示系统正常
        ADGO=1；                      //启动下一次 A/D 转换
}
//主程序
main()
{
        adinitial()；                 //A/D 转换初始化
        initial()；                   //系统各 I/O 口初始化
        ei()；                        //总中断允许
        ADGO=1；                      //启动 A/D 转换
while(1)
        {
                ；
        }                             //等待中断，在中断中循环检测外部电压
}
```

2.2.2 I²C 总线工作方式相关子程序

1．C 语言编写的 I²C 总线工作方式的初始化子程序

```
//I2C 初始化子程序
void        i2cint()
{
        SSPCON = 0X08；                   //初始化 SSPCON 寄存器
        TRISC3 =1；                   //设置 SCL 为输入口
        TRISC4 =1；                   //设置 SDA 为输入口
        TRISA4 = 0；
        SSPSTAT=0X80；                //初始化 SSPSTAT 寄存器
        SSPADD=0X02；                 //设定 I2C 时钟频率
        SSPCON2=0X00；                    //初始化 SSPCON2 寄存器
        di()；                        //关闭总中断
        SSPIF=0；                     //清 SSP 中断标志
        RA4=0；                       //关掉 74HC165 的移位时钟使能，以免 74HC165 移位
```

```
                                    //数据输出与I2C总线的数据线发生冲突（此操作与该
                                    //实验板的特殊结构有关，不是通用的）
    SSPEN=1；                       //SSP 模块使能
}
```

2．C 语言编写的 I^2C 总线工作方式传输数据子程序

需要发送的数据在寄存器 j 中。

```
//I2C 总线输出数据子程序
i2cout()
{
    SEN=1；                         //产生 I2C 启动信号
    for(n=0x02；--n；)    continue；//给予一定的延时，保证启动
do  {
        RSEN=1；                    //产生 I2C 重启动信号
    }while(SSPIF==0)；              //如果没能启动，则反复启动，直到启动为止
    SSPIF=0；                       //SSPIF 标志清 0
    SSPBUF=0X58；                   //I2C 总线发送地址字节
do  {
        ；
    }while(SSPIF==0)；              //等待地址发送完毕
    SSPIF=0；                       //SSPIF 标志清 0
    SSPBUF=0X01；                   //I2C 总线发送命令字节
do  {
        ；
    }while(SSPIF==0)；              //等待命令发送完毕
    SSPIF=0；                       //SSPIF 标志清 0
    SSPBUF=j；                      //I2C 总线发送数据字节
do  {
        ；
    }while(SSPIF==0)；              //等待数据发送完毕
    SSPIF=0；                       //SSPIF 标志清 0
    PEN=1；                         //产生停止条件
do {
        ；
    }while(SSPIF==0)；              //等待停止条件产生
    SSPIF=0；                       //SSPIF 标志清 0
}
```

2.2.4 程序清单

下面给一个例程。该程序利用 MAX518 进行 D/A 转换，且从 D/A0 引脚输出一个正弦波形。可作为读者编制程序的参考。特别注意，在调试该程序时，把模板上的钮子开关 S8 拔向高电平，以免发生资源冲突。

```
    #include  <pic.h>
//本程序将通过 PIC16F877 的 I2C 方式驱动 D/A 转换器 MAX518，使其 D/A0 通道输出
//一个连续的正弦波形（注：本程序并没对正弦波的频率进行控制）
const   char   table[ ] = {0X80，0X86，0X8D，  0X93，0X99，0X9F，0XA5，0XAB，
0XB1，0XB7，0XBC，0XC2，0XC7，0XCC，0XD1，0XD6，0XDA，0XDF，0XE3，0XE7，
0XEA，0XEE，0XF1，0XF4，0XF6，0XF8，0XFA，0XFC，0XFD，0XFF，0XFF，0XFF，
0XFF，0XFF，0XFF，0XFE，0XFD，0XFB，0XF9，0XF7，0XF5，0XF2，0XEF，0XEC，
0XE9，0XE5，0XE1，0XDD，0XD8，0XD4，0XCF，0XCA，0XC5，0XBF，0XBA，0XB4，
0XAE，0XA8，0XA2，0X9C，0X96，0X90，0X89，0X83，0X80，0X79，0X72，0X6C，
0X66，0X60，0X5A，0X55，0X4E，0X48，0X43，0X3D，0X38，0X33，0X2E，0X29，
0X25，0X20，0X1C，0X18，0X15，0X11，0X0E，0X0B，0X09，0X07，0X05，0X03，0X02，
```

```
0X00，0X00，0X00，0X00，0X00，0X00，0X01，0X02，0X04，0X06，0X08，0X0A，0X0D，
0X10，0X13，0X16，0X1A，0X1E，0X22，0X27，0X2B，0X30，0X35，0X3A，0X40，
0X45，0X4C，0X51，0X57，0X5D，0X63，0X69，0X6F，0X76，0X7C}；
//以上的数组用于存放正弦表，在定义数组时，前面应该加上 const，
//以使数组存放于 ROM 中，而不至于占用太多的 RAM
unsigned    char   i；
unsigned        char      j；
unsigned        char      n；
//I2C 初始化子程序
void    i2cint()
{
    SSPCON = 0X08；             //初始化 SSPCON 寄存器
    TRISC3 =1；             //设置 SCL 为输入口
    TRISC4 =1；             //设置 SDA 为输入口
    TRISA4 = 0；
    SSPSTAT=0X80；          //初始化 SSPSTAT 寄存器
    SSPADD=0X02；           //设定 I2C 时钟频率
    SSPCON2=0X00；              //初始化 SSPCON2 寄存器
    di()；                  //关闭总中断
    SSPIF=0；               //清 SSP 中断标志
    RA4=0；                 //关掉 74HC165 的移位时钟使能，以免 74HC165
                            //移位数据输出与 I2C 总线的数据线发生冲突
    SSPEN=1；               //SSP 模块使能
}
//I2C 总线输出数据子程序
void    i2cout()
{
    SEN=1；                 //产生 I2C 启动信号
    for(n=0x02；--n；)    continue；//给予一定的延时，保证启动
do  {
        RSEN=1；            //产生 I2C 启动信号
    }while(SSPIF==0)；      //如果没能启动，则反复启动，直到启动为止
    SSPIF=0；               //SSPIF 标志清 0
    SSPBUF=0X58；           //I2C 总线发送地址字节
do  {
        ；
    }while(SSPIF==0)；      //等待地址发送完毕
    SSPIF=0；               //SSPIF 标志清 0
    SSPBUF=0X01；           //I2C 总线发送命令字节
do  {
        ；
    }while(SSPIF==0)；      //等待命令发送完毕
    SSPIF=0；               //SSPIF 标志清 0
    SSPBUF=j；              //I2C 总线发送数据字节
do  {
        ；
    }while(SSPIF==0)；      //等待数据发送完毕
    SSPIF=0；               //SSPIF 标志清 0
    PEN=1；                 //产生停止条件
do  {
        ；
    }while(SSPIF==0)；      //等待停止条件产生
```

```
    SSPIF=0；                //SSPIF 标志清 0
}
//主程序
main ()
{
    i2cint()；               //I2C 初始化
while(1){
        for(i=0x00；i<=127；++i)
        {
            j=table[i]；     //从数组中得到需要传输的数据量
            i2cout()；       //利用 I2C 总线方式送出数据
        }
    }
}
```

NOTE：

第3章　秒　表

3.2.2　程序清单

该源程序已在实验板上调试通过，读者可直接引用，并可利用软件编程的灵活性，加以拓展，实现更为复杂的功能。

```
#include  <pic.h>
#include  <math.h>
//此程序实现计时秒表功能，时钟显示范围 00.00～95.99 秒，分辨度:0.01 秒
unsigned   char        s0，s1，s2，s3；
//定义 0.01  秒、0.1  秒、1 秒、10 秒计时器
unsigned   char    s[4]；
unsigned   char     k ，data ，sreg；
unsigned    int        i；
const      table[10]={0xc0，0xf9，0xa4，0xb0，0x99，0x92，0x82，0XD8，0x80，0x90}；
//不带小数点的显示段码表
const      table0[10]={0X40，0X79，0X24，0X30，0X19，0X12，0X02，0X78，0X00，0X10}；
//带小数点的显示段码表
//TMR0 初始化子程序
void       tmint()
{
     T0CS=0；                    //TMR0 工作于定时器方式
     PSA=1；                     //TMR0 不用分频
     T0IF=0；                        //清除 TMR0 的中断标志
     T0IE=1；                        //TMR0 中断允许
}
//spi 显示初始化子程序
void       SPIINIT()
{
     PIR1=0；
     SSPCON=0x30；
     SSPSTAT=0xC0；
//设置 SPI 的控制方式，允许 SSP 方式，并且时钟下降沿发送。与"74HC595，当其
//SCLK 从低到高跳变时，串行输入寄存器"的特点相对应
     TRISC=0xD7；                //SDO 引脚为输出，SCK 引脚为输出
     TRISA5=0；                  //RA5 引脚置为输出，输出显示锁存信号
}
//系统其它部分初始化子程序
void       initial()
{
     TRISB1=0；
     TRISB2=0；
     TRISB4=1；
     TRISB5=1；                  //设置与键盘有关的各口的输入输出方式
     RB1=0；
     RB2=0；                     //建立键盘扫描的初始条件
}
//SPI 传输数据子程序
void       SPILED(data)
{
     SSPBUF=data；               //启动发送
```

```
    do  {
            ;
        }while(SSPIF==0)；
    SSPIF=0；
}
//显示子程序，显示 4 位数
void    dispaly()
{
    RA5=0；                     //准备锁存
    for(k=4；k>0；k--)
    {
        data=s[k-1]；
        if(k==3)  data=table0[data]；//第二位需要显示小数点
        else  data=table[data]；
        SPILED(data)；          //发送显示段码
    }
    for(k=0；k<4；k++)
    {
        data=0xFF；
        SPILED(data)；          //连续发送 4 个 DARK，使显示好看一些
    }
    RA5=1；                     //最后给锁存信号，代表显示任务完成
}
//软件延时子程序
void        DELAY()
{
    for(i = 3553；  --i  ；)  continue；
}
//键扫描子程序
void    KEYSCAN()
{
while(1){
while(1)
    {
        dispaly()；             //调用一次显示子程序
        if ((RB5==0)||(RB4==0))     break；
    }
        DELAY()；               //若有键按下，则软件延时
        if ((RB5==0)||(RB4==0))     break；//若还有键按下，则终止循环扫描，返回
    }
}
//等键松开子程序
void    keyrelax()
{
while(1){
    dispaly()；                 //调用一次显示子程序
    if ((RB5==1)&&(RB4==1))  break；
    }                           //为防止按键过于灵敏，每次等键松开才返回
}
//系统赋值初始化子程序
void    inizhi()
{
    s0=0x00；
    s[0]=s0；
```

```
        s1=0x00；
        s[1]=s1；
        s2=0x00；
        s[2]=s2；
        s3=0x00；
        s[3]=s3；                          //s0=s1=s2=s3=0，并放入显示缓冲数组中
        sreg=0x00；                    //tmr0 中断次数寄存器清 0
}
//中断服务程序
void       interrupt  clkint(void)
{
        TMR0=0X13；                    //对 TMR0 写入一个调整值。因为写入 TMR0 后接着的
                                       //两个周期不能增量，中断需要 3 个周期的响应时间，
                                       //以及 C 语言自动进行现场保护要消耗周期
        T0IF=0；                           //清除中断标志
        CLRWDT()；
        sreg=sreg+1；                  //中断计数器加 1
        if(sreg==40)                   //中断次数为 40 后，才对 S0，S1，S2，S3  操作
        {
                sreg=0；
                s0=s0+1；
                if(s0==10){
                        s0=0；
                        s1=s1+1；
                        if(s1==10){
                                s1=0；
                                s2=s2+1；
                                if(s2==10){
                                        s2=0；
                                        s3=s3+1；
                                        if(s3==10)      s3=0；
                                }
                        }
                }
        }
        s[0]=s0；
        s[1]=s1；
        s[2]=s2；
        s[3]=s3；
}
//主程序
main()
{
        OPTION=0XFF；
        tmint()；                      //TMR0 初始化
        SPIINIT()；                    //spi 显示初始化
        initial()；                    //系统其它部分初始化
        di()；                         //总中断禁止
        while(1)  {
                inizhi()；             //系统赋值初始化
                KEYSCAN()；            //键扫描，直到开始键按下
                keyrelax()；           //等键松开
```

```
            ei()；                     //总中断允许
            KEYSCAN()；                //键扫描直到停止键按下，在键扫描时有显示
            keyrelax()    ；                //等键松开
            di()；                     //总中断禁止
            KEYSCAN()；                //键扫描到清 0 键按下，在键扫描时有显示
            keyrelax()    ；                //等键松开
      }
}
```

NOTE：

第 4 章　通用同步/异步通信的应用

4.1 单片机双机异步通信

4.1.1　单片机 PIC1 编程（发送部分）

```
#include  <pic.h>
/*该程序实现单片机双机异步通信功能，该程序是发送部分*/
unsigned char tran[8]；        /*定义一个数组存储发送数据*/
unsigned char k，data；       /*定义通用寄存器*/
const   char   table[20]={0xc0，0xf9，0xa4，0xb0，0x99，0x92，0x82，0XD8，0x80，0x90，
0x88，0x83，0xc6，0xa1，0x86，0x8e，0x7f，0xbf，0x89，0xff}；
/*不带小数点的显示段码表*/
/*spi 显示初始化子程序*/
void        SPIINIT()
{
        PIR1=0；
        SSPCON=0x30；
        SSPSTAT=0xC0；
/*设置 SPI 的控制方式，允许 SSP 方式，并且时钟下降沿发送，与"74HC595，当其
*SCLK 从低到高跳变时，串行输入寄存器"的特点相对应*/
        TRISC=0xD7；                  /*SDO 引脚为输出，SCK 引脚为输出*/
        TRISA5=0；                    /*RA5 引脚设置为输出，以输出显示锁存信号*/
}
/*给数组赋初值子程序  */
void        fuzhi()
{
        for(k=0；k<8；k++)     {
                tran[k]=k+3；
        }
}
/*SCI 部件初始化子程序*/
void        sciint()
{
        SPBRG=0X19；                  /*将传输的波特率设为约 9 600 位/秒*/
        TXSTA=0X04；                  /*选择异步高速方式传输 8 位数据*/
        RCSTA=0X80；                  /*允许同步串行口工作*/
        TRISC6=1；
        TRISC7=1；                    /*将 RC6、RC7 设置为输入方式，对外部呈高阻状态*/
}
/*SPI 传输数据子程序*/
void        SPILED(data)
{
        SSPBUF=data；                 /*启动发送*/
        do    {
                ；
        }while(SSPIF==0)；
        SSPIF=0；
}
/*显示子程序，显示 8 位数*/
void        display()
{
        RA5=0；                       /*准备锁存*/
```

```
    for(k=0；k<8；k++)    {
        data=tran[k]；
        data=table[data]；        /*查得显示的段码*/
        SPILED(data)；            /*发送显示段码*/
    }
    RA5=1；                       /*最后给一个锁存信号，代表显示任务完成*/
}
/*主程序*/
main()
{
    SPIINIT()；
    fuzhi()；                     /*给数组赋初值*/
    sciint()；                    /*SCI 部件初始化*/
    di()；                        /*中断禁止*/
    TXEN=1；                      /*发送允许*/
    CREN=1；                      /*接收数据允许*/
    for(k=0；k<8；k++){
        TXREG=tran[k]；           /*发出一个字符*/
        while(1){
            if(TXIF==1) break；
        }                         /*等待写入完成*/
        while(1){
            if(RCIF==1)   break；/*若收到响应字节，则终止等待*/
        }
        RCREG=RCREG；             /*读响应字节，清 RCIF*/
     }
    display()；                   /*显示发送的数据*/
    while(1){
        ；
    }
}
```

4.1.2 单片机 PIC2 编程（接收部分）

```
#include  <pic.h>
/*该程序实现单片机双机异步通信功能，该程序是接收部分，并把接收的数据显示在 8
*个 LED 上*/
unsigned    char        rece[8]；/*定义一个数组存储接收数据*/
unsigned    char       k，data；/*定义通用寄存器*/
const       char        table[20]={0xc0，0xf9，0xa4，0xb0，0x99，0x92，0x82，0XD8，0x80，
0x90，0x88，0x83，0xc6，0xa1，0x86，0x8e，0x7f，0xbf，0x89，0xff}；
/*不带小数点的显示段码表*/
/*spi 显示初始化子程序*/
void      SPIINIT()
{
     ；详细语句见发送程序
}
/*SCI 部件初始化子程序*/
void      sciint()
{
    SPBRG=0X19；                  /*波特率设置与 PIC1 相同，为约 9 600 位/秒*/
    TXSTA=0X04；                  /*异步高速传输*/
    RCSTA=0X80；                  /*串行口工作使能*/
    TRISC6=1；
```

```
    TRISC7=1；               /*将 RC6、RC7 设置为输入方式，对外部呈高阻状态*/
}
/*SPI 传送数据子程序*/
void      SPILED(data)
{
     ；详细语句与见发送程序
}
/*显示子程序，显示 4 位数*/
void display()
{
    RA5=0；                  /*准备锁存*/
    for(k=0；k<8；k++){
        data=rece[k]；
        data=table[data]；   /*查得显示的段码*/
        SPILED(data)；       /*发送显示段码*/
    }
    RA5=1；                  /*最后给一个锁存信号，代表显示任务完成*/
}
/*主程序*/
main()
{
    SPIINIT()；              /*spi 显示初始化*/
    sciint()；               /*SCI 部件初始化*/
    di()；                   /*中断禁止*/
    CREN=1；                 /*接收允许*/
    TXEN=1；                 /*发送允许*/
    for(k=0；k<8；k++){
        while(1){
            if(RCIF==1)  break；
        }                    /*等待接收数据*/
        rece[k]=RCREG；      /*读取接收数据，同时清掉 RCIF*/
        TXREG=rece[k]；      /*发送接收到的数据*/
        while(1){
            if(TXIF==1)  break；
        }                    /*等待写入完成*/
    }
    display()；              /*显示接收的数据*/
    while(1){
        ；
    }
}
```

4.2 单片机双机同步通信

4.2.1 单片机 PIC1 编程（主控发送）

```
#include <pic.h>
/*该程序实现单片机双机同步通信功能，是主控发送部分。程序上电后显示
*相应的字符，表示系统正常工作。发送完毕后显示发送的数据*/
unsigned    char    tran[8]；     /*定义一个数组存储发送数据*/
unsigned    char    k，data；     /*定义通用寄存器*/
const       char    table[20]={0xc0，0xf9，0xa4，0xb0，0x99，0x92，0x82，0XD8，0x80，
0x90，0x88，0x83，0xc6，0xa1，0x86，0x8e，0x7f，0xbf，0x89，0xff}；
/*不带小数点的的显示段码表*/
/*spi 显示初始化子程序*/
```

```
void     SPIINIT()
{
    ；详细程序语句请参考本章 4.5 节
}
/*给发送数组赋初值子程序 */
void     fuzhi()
{
    for(k=0；k<8；k++){
        tran[k]=k；
    }                   /*发送 0～7 八个数据*/
}
/*SCI 部件初始化子程序*/
void     sciint()
{
    SPBRG=200 ；        /*将传输的波特率设为约 9600 位/秒*/
    TXSTA=0X90；        /*选择主控方式*/
    RCSTA=0X80；        /*允许同步串行口工作*/
    TRISC6=1；
    TRISC7=1；          /*将 RC6、RC7 设置为输入方式，对外部呈高阻状态*/
}
/*SPI 传送数据子程序*/
void     SPILED(data)
{
    ；详细程序语句请参考本章 4.5 节
}
/*显示子程序，显示 8 位数*/
void     display()
{
    RA5=0；             /*准备锁存*/
    for(k=0；k<8；k++){
        data=tran[k]；
        data=table[data]；  /*查得显示的段码*/
        SPILED(data)；   /*发送显示段码*/
    }
    RA5=1；             /*最后给一个锁存信号，代表显示任务完成*/
}
/*显示子程序，显示 8 位数*/
void     display1()
{
    RA5=0；             /*准备锁存*/
    for(k=0；k<8；k++){
        data=0xf9；      /*显示"1"表示系统正常工作*/
        SPILED(data)；   /*发送显示段码*/
    }
    RA5=1；             /*最后给一个锁存信号，代表显示任务完成*/
}
/*主程序*/
main()
{
    SPIINIT()；             /*spi 显示初始化*/
    fuzhi()；               /*给发送数组赋发送初值*/
    sciint()；              /*SCI 部件初始化*/
    di()；                  /*中断禁止*/
```

```
    TXEN=1；                    /*发送允许*/
    display1()；                /*显示相应的字符，表示系统正常*/
    while(1){
        for(k=0；k<8；k++){
            TXREG=tran[k]；/*发出一个字符*/
            while(1){
                if(TXIF==1)  break；
            }                   /*等待上一个数据写入完成*/
        }
        display()；             /*显示发送的数据*/
    }                           /*循环发送*/
}
```

4.2.2 单片机 PIC2 编程（从动接收）

```
#include     <pic.h>
/*该程序实现单片机双机 同步通信功能，是从动接收部分，并把接收的数据显
*示在 8 个 LED 上*/
unsigned     char     rece[8]；       /*定义一个数组存储接收数据*/
unsigned     char     k，data；       /*定义通用寄存器*/
unsigned     int      i；
const   char   table[20]={0xc0，0xf9，0xa4，0xb0，0x99，0x92，0x82，0XD8，0x80，0x90，
0x88，0x83，0xc6，0xa1，0x86，0x8e，0x7f，0xbf，0x89，0xff}；
/*不带小数点的显示段码表*/
/*spi 显示初始化子程序*/
void     SPIINIT()
{
    ；详细程序语句请参考本章 4.5 节
}
/*SCI 部件初始化子程序*/
void     sciint()
{
    TXSTA=0X10；                /*选择同步从动方式*/
    RCSTA=0X90；                /*串行口工作使能*/
    TRISC6=1；
    TRISC7=1；                  /*将 RC6、RC7 设置为输入方式对外部呈高阻状态*/
}
/*SPI 传送数据子程序*/
void     SPILED(data)
{
    ；/*详细程序语句请参考本章 4.5 节*/
}
/*显示子程序，显示 4 位数*/
void     display()
{
    RA5=0；                     /*准备锁存*/
    for(k=0；k<8；k++){
        data=rece[k]；
        data=table[data]；      /*查得显示的段码*/
        SPILED(data)；          /*发送显示段码*/
    }
    RA5=1；                     /*最后给一个锁存信号，代表显示任务完成*/
}
/*主程序*/
```

```
main()
{
    SPIINIT()；                   /*spi 显示初始化*/
    sciint()；                    /*SCI 部件初始化*/
    di()；                        /*中断禁止*/
    CREN=1；                      /*接收允许*/
    for(k=0；k<8；k++) rece[k]=0x03；
    display()；                   /*显示表示系统正常运行的数据*/
    while(1)  {
        while(1){
            CREN=1；              /*允许连续接收*/
            while(1){
                if(RCIF==1)   break；
            }                     /*等待接收数据*/
            k=0；
            rece[k]=RCREG；  /*读取接收数据*/
            if(OERR==1) {         /*如果有溢出错误,则处理*/
                CREN=0；
                CREN=1；
            }
    if(rece[k]==0x00) break；/*“0”为同步字符,只有接收到“0”时才进行下面的接收*/
        }
        for(k=1；k<8；k++){
            while(1){
                if(RCIF==1)   break；
            }                     /*等待接收数据*/
            rece[k]=RCREG；/*读取接收数据*/
            if(OERR==1)  {        /*如果有溢出错误,则处理*/
                CREN=0；
                CREN=1；
            }
            rece[k]=rece[k]&0x0F；/*屏蔽掉高位,防止干扰*/
        }
        CREN=0；
        display()；                       /*显示接收的数据*/
        for(i=65535；--i；  )continue；
        for(i=65535；--i；  )continue；/*给予一定时间的延时，再进行下一轮接收*/
    }
}
```

4.3 单片机与 PC 机通信

4.3.1 PC 机编程

PC 采用 Toubr C 进行编写。程序如下：

```
#include<stdio.h>
#define      port     0x3f8                  /*利用串口 1 进行通信*/
int ch[15]；
main ()
{
   int a；
   int i，j；
   int b[6]={88，15，38，26，20，0}；
   char c；
```

```
clrscr()；
outportb(port+3，0x80)；          /*准备设置波特率*/
outportb(port，0x0C)；            /*波特率设置为 9600bps*/
outportb(port+1，0x00)；
outportb(port+3，0x03)；          /*8 位数据，无奇偶检验，1 位停止位*/
outportb(port+1，0x00)；          /*关中断*/
inportb(port+5)；                 /*读一次线路状态寄存器，使其复位*/
for(;;){
  printf("\t\tsend data or receive data: (s or r?)\n\n\n")；
  c=getchar()；
   switch(c) {
     case 's':
     case 'S': {
       while(!(inportb(port+5)&0x20))；/*发送保持器满则等待*/
       outportb(port，0x01)；         /*否则发送数据 01，通知单片机准备接收*/
       for(i=0；i<6；i++){            /*共发送 6 个数据*/
        a=b[i]；
        while(!(inportb(port+5)&0x20)) delay(100)；/*发送保持器满，等待*/
        outportb(port，a)；           /*发送 a*/
        printf("%d\n"，a)；           /*显示 a*/
        while(!(inport(port+5)&1))；  /*接收单片机送回的数据*/
        ch[i]=inport(port)；          /*保存*/
    }
    delay(10)；
    for(j=0；j<8；j++)   printf("\n%d\n"，ch[j])；/*显示接收的回送数据*/
    getch()；
    break；
  }
  case'r':                            /*接收数据*/
  case'R':{
      while(!(inportb(port+5)&0x20))；
       outportb(port，0x02)；         /*发送数据 02，通知单片机发送数据*/
       for(j=0；j<9；j++){            /*共接收 9 个数据*/
        while(!(inportb(port+5)&1))；
       ch[j]=inportb(port)；
       }
       for(j=0；j<9；j++) printf("\n %d\n"，ch[j])；
       getch()；
       break；
       }
    }
  }
}
```

NOTE：

第5章　PIC16F87X 在 CAN 通信中的应用

5.1　软件清单

```
// ========CAN 通信程序=======
#include    <pic.h>
#include    <pic16f87x.h>
#include    <mcp256.h>          // MCP2510 寄存器定义
// =========常数和变量定义=========
#define READ    0x03                // 读 MCP2510 指令代码
#define WRITE   0x02            // 写 MCP2510 指令代码
#define RESET   0xC0            // 复位 MCP2510 指令代码
#define RTS 0x80                // MCP2510 请求发送指令代码
#define STA2510 0xA0            // 读 MCP2510 状态指令代码
#define BITMOD  0x05            // MCP2510 位修改指令代码
int a[12];                  // SPI 发送或接收数据寄存器
int b[8];                   // 发送或接收的数据
int c[8];                   // 发送或接收的数据
int i;                      // 临时变量
int count;                  // 发送接收计数器
int count1=0;               // for test
int RecID_H=0;
int RecID_L=0;
int DLC=8;
void SPIINT();
void TMR1INT();
void CCP1INT();
void SPIEXCHANGE(int count);
void WAIT_SPI();
void RESET2510();
int  RD2510(int adress,int n);
void WR2510(int adress,int n);
void RTS2510(int RTSn);
int  GETS2510();
void BM2510(int adress,int mask,int data);
void SETNORMAL();
void TXCOMPLETE(int adress);
void TXMSG(int DLC);
int  RXMSG();
void INIT2510();
void INIT877();
void INITSPI();
void ACK();
void wait();
// ========主程序=======
main(void)
{
    int l,detect=0;
    SSPIE=1;
    TMR1IE=1;
```

```
    CCP1IE=1;
    CCP2IE=1;
    PEIE=1;
    ei();                   // 开中断
    INIT877();              // 初始化 PIC16F877 芯片
    INITSPI();              // 初始化 SPI 接口
    INIT2510();             // 初始化 MCP2510 芯片
    flag1=0;
    flag2=0;
    CCP1CON=0x05;
    CCP2CON=0x04;
    while(1)    {
        RXMSG();
        TXMSG(8);
    }
}
// ========中断服务程序=======
// SPI 中断服务子程序
void SPIINT()
{
    SSPIF=0;
    a[i++]=SSPBUF;          // 数据暂存 a[]中
    count-=1;
    if(count>0)  SSPBUF=a[i];// 未发送完，继续
    else  RE2=1;                // 否则，片选信号置高电平
    return;
}
// TMR1 中断服务子程序
void TMR1INT()
{
    TMR1IF=0;
    T1CON=0;
    if(!flag1){
        TMR1H=0xfe;             // 512 μs 脉冲宽度
        TMR1L=0x00;
        T1CON=0x01;
        PORTD=0xff;             // 输出所有通道
        flag1=1;
    }
    else    {
        flag1=0;
        PORTD=0;
        T1CON=0;
    }
    return;
}
// CCP1 中断服务子程序
void CCP1INT()
{
    CCP1IF=0;
    T1CON=0x01;
```

```
    return;
}
// CCP2 中断服务子程序
 void CCP2INT()
{
    CCP2IF=0;
    T1CON=0x01;
    return;
}
// 中断入口，保护现场，判中断类型
void interrupt INTS()
{
    di();
    if(TMR1IF)  TMR1INT();      // 定时器 TMR1 中断
    else if(CCP1IF)  CCP1INT(); // 电压过零捕捉中断 1
    else if(CCP2IF)  CCP2INT(); // 电压过零捕捉中断 2
    else if(SSPIF)  SPIINT();       // SPI 接口中断
    ei();
}
// ========子程序=======
// 启动 SPI 传送
 void SPIEXCHANGE(count)
 int count;
{
    if(count>0) {               // 有数据可送？
      i=0;
      RE2=0;                        // 片选位置低电平
      SSPBUF=a[i];              // 送数
    }
    else
      ;                         // 否则，空操作，并返回
    return;
}
// 等待 SPI 传送完成
 void WAIT_SPI()
{
    do{
      ;
    }while(count>0);                // 当 count!=0 时，等待 to add "CLRWDT"
    return;
}
// 对 MCP2510 芯片进行复位
void RESET2510()
{
    a[0]=RESET;
    count=1;
    SPIEXCHANGE(count);         // 送复位指令
    WAIT_SPI();
    return;
}
// 读取从地址"adress"开始的寄存器中的数据，共 n 个，存放在数组 b[n]中
```

```
 int RD2510(adress,n)
 int    adress;
 int        n;
{
    int j;
    a[0]=READ;
    a[1]=adress;
    for(j=0;j<n;j++)  a[j+2]=0;
    count=n+2;                  // 指令、地址和要得到的数据量 n
    SPIEXCHANGE(count);
    WAIT_SPI();
    for(j=0;j<n;j++)  b[j]=a[j+2];// 数据存到数组 b[]中
    return;
}
// 向从地址"adress"开始的寄存器写入数据，共 n 个，数据存放数组 b[n]中
 void WR2510(adress,n)
 int        adress;
 int        n;
{
    int j;
    a[0]=WRITE;
    a[1]=adress;
    for(j=0;j<n;j++) a[j+2]=b[j];
    count=n+2;                  // 指令、地址和要写入的数据量 n
    SPIEXCHANGE(count);
    WAIT_SPI();
    return;
}
// MCP2510 芯片请求发送程序
 void RTS2510(RTSn)
 int RTSn;
{
    a[0]=RTS^RTSn;
    count=1;
    SPIEXCHANGE(count);         // 发送 MCP2510 芯片，请求发送指令
    WAIT_SPI();
    return;
}
// 读取 MCP2510 芯片的状态
 int GETS2510()
{
    a[0]=STA2510;
    a[1]=0;
    count=2;
    SPIEXCHANGE(count);         // 读取 MCP2510 芯片状态
    WAIT_SPI();
    b[0]=a[1];                  // 状态存到数组 b[]中
    return;
}
// 对 MCP2510 芯片进行位修改子程序
 void BM2510(adress,mask,data)
```

```
 int  adress;
 int  mask;
 int  data;
{
    a[0]=BITMOD;                    // 位修改指令
    a[1]=adress;                    // 位修改寄存器地址
    a[2]=mask;                  // 位修改屏蔽位
    a[3]=data;                  // 位修改数据
    count=4;
    SPIEXCHANGE(count);
    WAIT_SPI();
    return;
}
// 设置 MCP2510 芯片为正常操作模式
void  SETNORMAL()
{
    int  k=1;
    BM2510(CANCTRL,0xe0,0x00);  // 设置为正常操作模式
    do  {
      RD2510(CANSTAT,1);
      k=b[0]&0xe0;
    }while(k);                  // 确认已进入正常操作模式
    return;
}
// 对 MCP2510 进行初始化
void INIT2510()
{
    RESET2510();                    // 使芯片复位
    b[0]=0x02;
    b[1]=0x90;
    b[2]=0x07;
    WR2510(CNF3,3);             // 波特率为 125 kbps
    b[0]=0x00;
    b[1]=0x00;
    WR2510(RXM0SIDH,2);
    b[0]=0x00;
    b[1]=0x00;
    WR2510(RXF0SIDH,2);         // RX0 接收，屏蔽位为 0，过滤器为 0
    b[0]=0x00;
    WR2510(CANINTE,1);          // CAN 中断不使能
    SETNORMAL();                    // 设置为正常操作模式
    return;
}
// MCP2510 芯片发送完成与否判断，邮箱号为 adress
void TXCOMPLETE(adress)
int adress;
{
    int k=1;
    do  {
      RD2510(adress,1);
      k=b[0]&0x08;
```

```
    }while(k);                  // 确认是否已发送完毕 to add CLRWDT
    return;
}
// 初始化 PIC16F877 芯片
void INIT877()
{
    PORTA=0;
    PORTB=0;
    PORTC=0;
    PORTD=0;
    PORTE=0;
    TRISA=0xff;
    TRISB=0xfd;
    TRISC=0xd7;                 // SCK, SDO:输出,SDI:输入
    TRISD=0;
    TRISE=0x03;                 // 片选 CS 信号输出
    PORTA=0xff;
    PORTB=0x03;                 // RST=1
    PORTC=0;
    PORTD=0xff;
    PORTE=0x04;
    return;
}
// 初始化 SPI 接口
 void INITSPI()
{
    SSPCON=0x11;
    SSPEN=1;                            // SSP 使能
    SSPSTAT=0;
    return;
}
// 发送数据子程序
 void  TXMSG(int DLC)
{
    for(i=0;i<DLC;i++)  b[i]=c[i];
    WR2510(TXB0D0,DLC);
    b[0]=DLC;
    WR2510(TXB0DLC,1);
    b[0]=0x03;
    b[1]=RecID_H;
    b[2]=RecID_L;
    WR2510(TXB0CTRL,3);
    RTS2510(0x01);                  // 请求发送
    TXCOMPLETE(TXB0CTRL);               //等待发送完毕
    return;
}
// 接收数据子程序
 int RXMSG()
{
    int k;
    RD2510(CANINTF,1);
```

```
    k=b[0]&0x01;
    if(k==1)    {
        BM2510(CANINTF,0x01,0x00);
        RD2510(RXB0SIDH,2);
        RecID_H=b[0];
        RecID_L=b[1]&0xe0;
        RD2510(RXB0DLC,1);
        DLC=b[0]&0x0f;
        RD2510(RXB0D0,DLC);
        for(i=0;i<DLC;i++) c[i]=b[i];
        return 1;
    }
    return 0;
}
```

NOTE：

第 6 章　利用 CCP 模块设计频率计

6.1　程序设计

6.1.1　程序清单

```
#include        <pic.h>
#include        <stdio.h>
#include        <math.h>
//本程序利用 CCP1 模块实现一个“简易数字频率计”的功能
const   char    table[11]={0xc0，0xf9，0xa4，0xb0，0x99，0x92，0x82，0XD8，0x80，0x90，
0xFF}；
//不带小数点的显示段码表
const   char    table0[11]={0X40，0X79，0X24，0X30，0X19，0X12，0X02，0X78，0X00，
0X10，0xFF}；
//带小数点的显示段码表
bank3   int   cp1z[11]；                //定义一个数组，用于存放各次的捕捉值
union   cp1
{int    y1；
    unsigned   char    cp1e[2]；
}cp1u；                                 //定义一个共用体
unsigned char COUNTW，COUNT；//测量脉冲个数寄存器
unsigned char COUNTER，data，k；
unsigned char FLAG @ 0XEF；
#define FLAGIT(adr，bit)   ((unsigned)(&adr)*8+(bit))   //绝对寻址位操作指令
static     bit FLAG1   @ FLAGIT(FLAG，0)；
static     bit FLAG2   @ FLAGIT(FLAG，1)；
static     bit FLAG3   @ FLAGIT(FLAG，2)；
unsigned char s[4]；                    //定义一个显示缓冲数组
int        T5  ，uo；
double     RE5；
double     puad5；
//spi 方式显示初始化子程序
void SPIINIT()
{
    PIR1=0；
    SSPCON=0x30；
    SSPSTAT=0xC0；
//设置 SPI 的控制方式，允许 SSP 方式，并且时钟下降沿发送，与"74HC595，当其
//SCLk 从低到高跳变时，串行输入寄存器"的特点相对应
    TRISC=0xD7；                       //SDO 引脚为输出，SCK 引脚为输出
    TRISA5=0；                         //RA5 引脚设置为输出，以输出显示锁存信号
    FLAG1=0    ；
    FLAG2=0    ；
    FLAG3=0    ；
    COUNTER=0X01；
}
//CCP 模块工作于捕捉方式初始化子程序
void ccpint( )
{
    CCP1CON=0X05；                         //首先设置 CCP1 捕捉每个脉冲的上升沿
    T1CON=0X00；                       //关闭 TMR1 震荡器
```

```
    PEIE=1；                          //外围中断允许(此时总中断关闭)
    CCP1IE=1；                        //允许 CCP1 中断
    TRISC2=1；                        //设置 RC2 为输入
}
//系统其它部分初始化子程序
void initial( )
{
    COUNT=0X0B；                      //为保证测试精度，测试 5 个脉冲的参数后
                                      //求平均值，每个脉冲都要捕捉其上升、下降沿，
                                      //故需要有 11 次中断
    TRISB1=0；
    TRISB2=0；
    TRISB4=1；
    TRISB5=1；                        //设置与键盘有关的各口的输入、输出方式
    RB1=0；
    RB2=0；                           //建立键盘扫描的初始条件
}
//SPI 传送数据子程序
void        SPILED(data)
{
    SSPBUF=data；                     //启动发送
    do   {
         ；
    }while(SSPIF==0)；
    SSPIF=0；
}
//显示子程序，显示 4 位数
void display( )
{
    RA5=0；                           //准备锁存
    for(COUNTW=0；COUNTW<4；COUNTW++){
        data=s[COUNTW]；
        data=data&0x0F；
        if(COUNTW==k)   data=table0[data]；//第二位需要显示小数点
        else  data=table[data]；
        SPILED(data)；                //发送显示段码
    }
    for(COUNTW=0；COUNTW<4；COUNTW++){
        data=0xFF；
        SPILED(data)；                //连续发送 4 个 DARK，使显示好看一些
    }
    RA5=1；                           //最后给一个锁存信号，代表显示任务完成
}
//键盘扫描子程序
void keyscan( )
{
    if((RB4==0)||(RB5==0))FLAG1=1     ；//若有键按下，则建立标志 FLAG1
    else  FLAG1=0     ；              //若无键按下，则清除标志 FLAG1
}
//键服务子程序
void     keyserve( )
{
    PORTB=0XFD     ；
```

```
    if(RB5==0)    data=0X01；
    if(RB4==0)    data=0X03；
    PORTB=0XFB；
    if(RB5==0)    data=0X02；
    if(RB4==0)    data=0X04；        //以上确定是哪个键按下
    PORTB=0X00；                     //恢复 PORTB 的值
    if(data==0x01)    {
        COUNTER=COUNTER+1；      //若按下 S9 键，则 COUNTER 加 1
        if(COUNTER>4)   COUNTER=0x01；//若 COUNTER 超过 4，则又从 1 计起
    }
    if(data==0x02)    {
        COUNTER=COUNTER-1；//若按下 S11 键，则 COUNTER 减 1
    if(COUNTER<1)   COUNTER=0x04；//若 COUNTER 小于 1，则又循环从 4 计起
    }
    if(data==0x03)    FLAG2=1    ；        //若按下 S10 键，则建立标志 FLAG2
    if(data==0x04)    FLAG2=0    ；        //若按下 S12 键，则清除标志 FLAG2
}
//中断服务程序
void  interruptcp1int(void)
{
    CCP1IF=0；                       //清除中断标志
    cp1u.cp1e[0]=CCPR1L；
    cp1u.cp1e[1]=CCPR1H；
    cp1z[data]=cp1u.y1；             //存储 1 次捕捉值
    CCP1CON=CCP1CON^0X01；   //把 CCP1 模块改变成捕捉相反的脉冲沿
    data++；
    COUNT--；
}
//周期处理子程序
void   PERIOD( )
{
    T5=cp1z[10]-cp1z[0]；            //求得 5 个周期的值
    RE5=(double)T5；                 //强制转换成双精度数
    RE5=RE5/5；                      //求得平均周期，单位为μs
}
//频率处理子程序
void   FREQUENCY( )
{
    PERIOD( )；                      //先求周期
    RE5=1000000/RE5；                //周期值求倒数，再乘以 1 000 000，得频率，
                                      //单位为 HZ
}
//脉宽处理子程序
void      PULSE( )
{
    int   pu；
    for(data=0，puad5=0；data<=9；data++)  {
        pu=cp1z[data+1]-cp1z[data]；
        puad5=(double)pu+puad5；
        data=data+2；
    }                                //求得 5 个脉宽的和值
    RE5=puad5/5；                    //求得平均脉宽
}
```

```
//占空比处理子程序
void    OCCUPATIONAL( )
{
    PULSE( )；                      //先求脉宽
    puad5=RE5；                     //暂存脉宽值
    PERIOD()；                      //再求周期
    RE5=puad5/RE5；                 //求得占空比
}
//主程序
main( )
{
    SPIINIT( )；                    //SPI 方式显示初始化
    while(1) {
        ccpint()；                      //CCP 模块工作于捕捉方式初始化
        initial()；                 //系统其它部分初始化
        if(FLAG2==0)    {
            s[0]=COUNTER；          //第一个存储 COUNTER 的值
            s[1]=0X0A；
            s[2]=0X0A；
            s[3]=0X0A；             //后面的 LED 将显示"DARK"
        }
        display( )；                //调用显示子程序
        keyscan()；                 //键盘扫描
        data=0x00；                 //存储数组指针赋初值
        TMR1H=0；
        TMR1L=0；                   //定时器 1 清 0
        CCP1IF=0；                  //清除 CCP1 的中断标志，以免中断一打开就进入
                                     //中断
        ei( )；                     //中断允许
        TMR1ON=1；                      //定时器 1 开
        while(1){
            if(COUNT==0)break；
        }                           //等待中断次数结束
        di()；                      //禁止中断
        TMR1ON=0；                      //关闭定时器
        keyscan()；                 //键盘扫描
        if(FLAG1==1)    keyserve()    ；  //若确实有键按下，则调用键服务程序

        if(FLAG2==0)    continue；      //如果没有按下确定键，则终止此次循环，
                                     //继续进行测量
    //如果按下了确定键，则进行下面的数值转换和显示工作
    if(COUNTER==0x01) FREQUENCY()；  //COUNTER=1，则需要进行频率处理
    if(COUNTER==0x02) PERIOD()；            //COUNTER=2，则需要进行周期处理
    if(COUNTER==0x03) OCCUPATIONAL()；//COUNTER=3，则需要进行占空比处理
    if(COUNTER==0x04) PULSE()；             //COUNTER=4，则需要进行脉宽处理
        k=5；
        if(RE5<1){
        RE5=RE5*1000；              //若 RE5<1，则乘以 1 000，保证小数点的精度
        k=0x00；
    }
    else  if(RE5<10){
        RE5=RE5*1000；              //若 RE5<10，则乘以 1 000，保证小数点的精度
```

```
            k=0x00；
        }
        else  if(RE5<100){
            RE5=RE5*100；                  //若 RE5<100，则乘以 100，保证小数点的精度
            k=0x01；
        }
        else  if(RE5<1000){
            RE5=RE5*10；                   //若 RE5<1000，则乘以 10，保证小数点的精度
            k=0x02；
        }
        else  RE5=RE5       ；
            uo=(int)RE5；
            sprintf(s，"%4d"，uo)；             //把需要显示的数据转换成 4 位 ASII 码,且放入数
                                           //组 S 中
            display()；
        }
}
```

NOTE：

第 7 章　交流电压测量

7.1.1　程序清单

该程序已在模板上调试通过，可作读者的参考。有关显示部分请读者参考本书相关章节，有关 A/D 转换的详细设置请参考前面章节。

```
#include        <pic.h>
#include        <math.h>
#include        <stdio.h>
//该程序用于测电网的交流电压有效值，最后的结果将在 4 个 LED 上显示，保留
//1 位小数。
//为了保证调试时数据运算的精确性，需要将 PICC 的 double 型数据选成 32 位
union    adres
{
    int   y1；
    unsigned      char adre[2]；
}adresult；                        //定义一个共用体
bank3    int   re[40]；            //定义存放 A/D 转换结果的数组，在 bank3 中
unsigned       char   k，data；        //定义几个通用寄存器
double    squ  ，squad；               //平方寄存器和平方和寄存器，squ 又通用为存储其
                                   //它数值
int   uo；
bank1    unsigned   char     s[4]；    //此数组用于存储需要显示的字符的 ASII 码
const         char    table[10]={0xc0，0xf9，0xa4，0xb0，0x99，0x92，0x82，0XD8，0x80，
0x90}；
//不带小数点的显示段码表
const         char    table0[10]={0x40，0x79，0x24，0x30，0x19，0x12，0x02，0x78，0x00，
0x10}；//带小数点的显示段码表
//A/D 转换初始化子程序
void       adinitial()
{
    ADCON0=0x41；                  //选择 A/D 通道为 RA0，且打开 A/D 转换器
                                   //在工作状态，使 A/D 转换时钟为 8Tosc
    ADCON1=0X8E；                  //转换结果右移，及 ADRESH 寄存器的高 6 位为"0"
                                   //把 RA0 口设置为模拟量输入方式
    ADIE=1；                       //A/D 转换中断允许
    PEIE=1；                           //外围中断允许
    TRISA0=1；                     //设置 RA0 为输入方式
}
//spi 方式显示初始化子程序
void                SPIINIT()
{
    PIR1=0；
    SSPCON=0x30；
    SSPSTAT=0xC0；
//设置 SPI 的控制方式，允许 SSP 方式，并且时钟下降沿发送，与"74HC595，当其
//SCLK 从低到高跳变时，串行输入寄存器"的特点相对应
    TRISC=0xD7；                   //SDO 引脚为输出，SCK 引脚为输出
    TRISA5=0；                     //RA5 引脚设置为输出，以输出显示锁存信号
}
//系统其它初始化子程序
```

```
void initial()
{
        CCP2IE=0；                      //禁止 CCP 中断
        SSPIE=0；                       //禁止 SSP 中断
        CCP2CON=0X0B；                  //初始化 CCP2CON，CCP2 为特别事件触发方式
        CCPR2H=0X01；
        CCPR2L=0XF4；                   //初始化 CCPR2 寄存器，设置采样间隔 500 μs，
                                        //一个周期内电压采 40 个点
}
//中断服务程序
void        interrupt        adint(void)
{
        CCP2IF=0；
        ADIF=0；                        //清除中断标志
        adresult.adre[0]=ADRESL；
        adresult.adre[1]=ADRESH；//读取并存储 A/D 转换结果，A/D 转换的结果
                                        //通过共用体的形式放入了变量 y1 中
        re[k]=adresult.y1；             //1 次 A/D 转换的结果存入数组
        k++；                           //数组访问指针加 1
}
//SPI 传送数据子程序
void        SPILED(data)
{
        SSPBUF=data；                   //启动发送
        do{
                ；
        }while(SSPIF==0)；
        SSPIF=0；
}
//主程序
main( )
{
        adinitial()；                   //A/D 转换初始化
        SPIINIT()；                     //spi 方式显示初始化
        initial()；                     //系统其它初始化
        while(1){
                k=0；                   //数组访问指针赋初值
                TMR1H=0X00        ；
                TMR1L=0X00；            //定时器 1 清 0
                ei()；                  //中断允许
                T1CON=0X01；            //打开定时器 1
                while(1){
                        if(k==40)break；  //A/D 转换次数达到 40，则终止
                }
                di()；                  //禁止中断
        for(k=0；k<40；k++)re[k]=re[k]-0X199；//假设提升电压为 2 V，对应十六进制数 199H，
                                        //则需在采样值的基础上减去该值
        for(k=0，squad=0；k<40；k++) {
                uo=re[k]；
                squ=(double)uo；        //强制把采得的数据量转换成双精度数，以便运算
                squ=squ*5/1023；        //把每点的数据转换成实际数据
                squ=squ*squ；                   //求一点电压的平方
```

```
        squad=squad+squu；
    }                               //以上求得 40 点电压的平方和，存于寄存器 squad 中
    squ=squad/40；                  //求得平均值
    squ=sqrt(squ)；                 //开平方，求得最后的电压值
    squ=squ*154.054；                  //通过变压器的变比和分压电阻分配确定该系数
                                    //以上得到了实际电网的电压值
    squ=squ*10；                    //为了保证显示的小数点的精度，先对电压值乘以 10
    uo=(int)squ；                   //强制把 U 转换成有符号整型量
    sprintf(s，"%4d"，uo)；            //通过 sprintf 函数把需要显示的电压数据转换成
                                    //ASII 码，并存于数组 S 中
    RA5=0；                         //准备锁存
    for(k=0；k<4；k++){
        data=s[k]；
        data=data&0X0F；            //通过按位相与的形式把 ASII 码转换成 BCD 码
        if(k==2)  data=table0[data]；//因为 squ 已乘以 10，则需在第 2 位打小数点
        else  data=table[data]；    // table0 存储带小数点的显示段码，
                                    //table 存储不带小数点的显示段码
        SPILED(data)；              //发送显示段码
    }
    for(k=0；k<4；k++)     {
        data=0xFF；
        SPILED(data)；      //连续发送 4 个 DARK，使显示看起来好看一些，这点与
                            //该实验板的 LED 分布结构有关
    }
    RA5=1；                 //最后给一个锁存信号，代表显示任务完成
    }
}
```

NOTE：

第 8 章　与 PLC 接口的 4 位 LED 数字显示表

8.1　数显表头软件设计思路

8.2　程序清单

```
#include    <pic16F87x.h>
#include    "mydefine.h"
#include    <pic.h>
static int flag,flag0,flag1,flag3,led_d;
static int data1[5],data2[5];
static int data,data0,data_1,data_2,sdata;
//=====================子程序========================
//端口初始化子程序
void initport( )
{
    PORTA=0;
    PORTB=0;
    PORTC=0;
    PORTD=0;
    ADCON1=0x07;
    TRISA=0x03;             //设 RA0,RA1 为输入
    TRISB=0xE8;             //设 RB0,RB1,RB2,RB4 为输出
    TRISC=0xFF;             //设 C 口为输入
    TRISD=0;                    //设 D 口为输出
}
//判断地址是否相同子程序
int adr_jud(int x)
{
    int adress,y;
    adress=PORTA&0x03;
    x&=0x60;
    adress=adress<<5;
    if (adress==x) y=1;
    else    y=0;
    CLRWDT();
    return(y);
}
//显示初始化子程序
void initdis( )
{
    PORTB=0xFE;             //选通数码管 1
    PORTD=0xC0;
    PORTB=0xFD;             //选通数码管 2
    PORTD=0xC0;
    PORTB=0xFB;             //选通数码管 3
    PORTD&=0x7F;                //选通小数位
    PORTD=0xC0;
    PORTB=0xEF;             //选通数码管 4
    PORTD=0xC0;
```

```
}
//读 5 次数据判是否有 4 次相等
int judge(arry)
int arry[5];
{
    int i,j,k;
    for(i=0;i<=4;i++){
        k=0;
        for(j=0;j<=4;j++)
          { if(arry[i]==arry[j])  k++;
            if(k>=4) {
              flag1=1;
              data0=arry[i];
              return(flag1);
           }
           else flag1=0;
         }
        }
    return(flag1);
}
//数据转换子程序
int convert(int d1,int d2)
{
    auto int dd1,dd2;
    int i1,j1,k1,i2,j2,m;
    dd1=d1;
    dd2=d2;
    j1=0x10;
    k1=2048;
    d1=0;
    for(i1=1;i1<=5;i1++) {
        if(j1==(dd1&j1)) m=1;
        else  m=0;
        d1=d1+m*k1;
        j1=j1/2;
        k1=k1/2;
      }
    j2=0x40;
    d2=0;
    for(i2=1;i2<=7;i2++) {
        if(j2==(dd2&j2))  m=1;
        else  m=0;
        d2=d2+m*k1;
        j2=j2/2;
        k1=k1/2;
     }
    data=d1+d2;
    return(data);
}
//显示子程序
int display(int x)
```

```
{   int l1,l2,l3,l4;
    l1=x/1000;
    PORTB=0xFE;             //选通数码管 1
    PORTD=led[l1];
    l2=(x-l1*1000)/100;
    PORTB=0xFD;             //选通数码管 2
    PORTD=led[l2];
    l3=(x-l1*1000-l2*100)/10;
    PORTB=0xFB;             //选通数码管 3
    PORTD=0x7F;
    PORTD=led[l3];
    l4=x-l1*1000-l2*100-l3*10;
    PORTB=0xEF;             //选通数码管 4
    PORTD=led[l4];
}
//中断服务子程序
void interrupt int_serve( )
{
    PIR1=0;
    TMR1L=0xE5;
    TMR1H=0xBE;
    di( );
    sdata=PORTC&0x80;
    ei( );
}
//开中断子程序
void int_open( )
{
    inportc=PORTC&0x80;
    if(inportc==1) return;
    else  data1[0]=~PORTC;
    flag=adr_jud(data1[0]);
    if(flag==0) return;     //地址不同返回
    else  data1[1]=~PORTC;
    data1[2]=~PORTC;
    if(data1[0]==data1[1])
       if(data1[0]==data1[2])   {
        flag3=1;
        PIR1=0;             //开通总中断前，清所有中断标志位
        TMR1IE=1;           //TMR1 溢出中断使能
        PEIE=1;
        ei( );
        TMR1L=0xE5;
        TMR1H=0xBE;         //20ms 中断 1 次
        T1CON=0x01;         //设 TMR1 为 1 分频,计数器方式工作
        }
    else  return;
}
//读第 1 帧子程序
voidread_1( )
{   int j0;
```

```
    for(j0=1;j0<=4;j0++)     data1[j0]=~PORTC;
    flag1=judge(data1);
    if (flag1==1) {
        data_1=data0;
        flag0=1;
        count1++;
    }
    flag=adr_jud(data1[0]);
    if(flag==1)  {
        for(j0=1;j0<=4;j0++) data1[j0]=~PORTC;
        flag1=judge(data1);
        if (flag1==1){
            data_1=data0;
            flag0=1;
            count1++;
        }
    }
}
// 主程序
main( )
{   int i0,ii,i;
    flag0=0;                    //帧标志位
    flag1=0;                    //读 5 次数据判有 4 次相等标志位
    flag3=1;                    //开中断标志位
    count1=0;               //读第 1 帧计数单元
    count2=0;               //读第 2 帧计数单元
    data_1=0;
    data_2=0;
    led_d=0;
    led[0]=0xc0;                //0
    led[1]=0xf9;
    led[2]=0xa4;
    led[3]=0xb0;
    led[4]=0x99;
    led[5]=0x92;
    led[6]=0x82;
    led[7]=0xf8;
    led[8]=0x80;
    led[9]=0x90;                //9
    initport( );
    OPTION=0xFE;                //开看门狗
    initdis( );
    while(1)    {
     if(flag3==0)  int_open();
     else{
        if(sdata==0x80){        //第二帧数据到
            if(flag0==1){
                for(i0=0;i0<=4;i0++)  data2[i0]=~PORTC;
                flag1=judge(data2);
                if (flag1==1)   {
                    data_2=data0;
```

```
                    flag0=0;
                    count2++;
                }
            }
        }
        else if(sdata==0) {         //第一帧数据到
            if(flag0==0)    {
                data1[0]=~PORTC;
                flag=adr_jud(data1[0]);
                if(flag==1) {
                    for(j0=1;j0<=4;j0++)     data1[j0]=~PORTC;
                        flag1=judge(data1);
                        if (flag1==1)   {
                        data_1=data0;
                        flag0=1;
                        count1++;
                     }
                  }
            }
        }
        CLRWDT();
        if(count1==count2)  led_d=convert(data_1,data_2 );
    }
    display(led_d);
  }
}
```

NOTE：

第9章　单片机控制的电动自行车驱动系统

9.1　C语言程序

```
#include    <pic.h>
//电动车双闭环程序，采用双闭环方式控制电机，以得到最好的 zh 转速性能，并且可以
//限制电机的最大电流。本应用程序用到两个 CCP 部件，其中 CCP1 用于 PWM 输出，以控
//制电机电压；CCP2 用于触发 AD，定时器 TMR2、TMR1，INT 中断，RB 口电平变化中断，
//看门狗以及 6 个通用 I/O 口
#define AND  0xe0       //状态采集 5，6，7 位
#define CURA 0X0a       //电流环比例和积分系数之和
#define CURB 0X09       //电流环比例系数
#define THL  0X6400     //电流环最大输出
#define FULLDUTY 0X0FF  //占空比为 1 时的高电平时间
#define SPEA 0X1d       //转速环比例和积分系数之和
#define SPEB 0X1c       //转速环比例系数
#define GCURHILO 0X0330 //转速环最大输出
#define GCURH 0X33      //最大给定电流
#define GSPEH 0X67      //最大转速给定
#define TSON 0X38       //手柄开启电压 1.1 V，TSON*2 为刹车后手柄开启电压，即
                         //2.2 V
#define VOLON 0X4c      //低电压保护重开电压 3.0 V 即 33 V
#define VOLOFF 0X49     //低电压保护关断电压 2.86 V 即 31.5 V
volatile unsigned char DELAYH,DELAYL,oldstate,speed,
    speedcount,tsh,count_ts,count_vol,gcur,currenth,
    voltage;                    //寄存器定义
static bit sp1,spe,ts,volflag,spepid,lowpower,
    off,shutdown,curpid;        //标志位定义
static volatile unsigned char new[10]={0xaf,0xbe,0xff,0x7e,0xcf,
    0xff,0xd7,0x77,0xff,0xff};  //状态寄存器表
//------------PIC16F877 初始化子程序------------
void INIT877()
{
    PORTC=0X0FF;            //关断所有 MOSFET
    TRISC=0X02;         //设置 C 口输出
    PIE1=0X00;          //中断寄存器初始化，关断所有中断
    TRISA=0XCF;         //设置 RA4,RA5 输出
    TRISB=0XEF;         //RB 口高三位输入，采集电机三相的霍尔信号
    PORTC=new[(PORTB&AND)>>5];  //采集第一次霍尔信号，并输出相应的信号，导通
                        //两个 MOS 管
    T2CON=0X01;         //TMR2 4 分频
    CCPR1L=0X0FF;       //初始时 PWM 输出全高
    CCP1CON=0X0FF;      //CCP1 设置为 PWM 方式
    CCP2CON=0X0B;       //CCP2 设置为特殊方式，以触发 AD
    ADCON0=0X81;            //AD 时钟为 32 分频,且 AD 使能,选择 AN0 通道采集手
                        //柄电压
    TMR2=0X00;          //TMR2 寄存器初始化
    TMR1H=0X00;         //TMR1 寄存器初始化
    TMR1L=0X00;
    T1CON=0X00;         //TMR1 为 1 分频
```

```
    CCPR2H=0X08;
    CCPR2L=0X00;            //电流采样周期设置为 TAD=512 μs
    PR2=0XC7;           //PWM 频率设置为 5 kHz
    ADCON1=0X02;            //AD 结果左移
    OPTION=0XFB;            //INT 上升沿触发
    TMR2ON=1;           //PWM 开始工作
    INTCON=0XD8;            //中断设置 GIE=1,PEIE=1,RBIE=1
    ADIE=1;             //AD 中断使能
    speedcount=0x00;        //转速计数寄存器
    speed=0x7f;         //转速保持寄存器
    spe=1;              //低速标志位
    sp1=1;              //低速标志位
    oldstate=0x0ff;     //初始状态设置，区别于其他状态
    count_ts=0x08;      //电流采样 8 次,采集 1 次手柄
    count_vol=0x00;     //采样 256 次手柄,采集 1 次电池电压
    ts=1;               //可以采集手柄值的标志位
    ADGO=1;             //AD 采样使能
    TMR1ON=1;           //CCP2 部件开始工作
}
//------------延时子程序---------------
#pragma interrupt_level 1
void DELAY1(x)
char x;
{
    DELAYH=x;           //延时参数设置
#asm
DELAY2  MOVLW   0X06
    MOVWF   _DELAYL
DELAY1  DECFSZ  _DELAYL
    GOTO    DELAY1
    DECFSZ  _DELAYH
    GOTO    DELAY2
#endasm
}
//-----------状态采集子程序----------------------
void sample()
{
    char state1,state2,state3,x;
    do  {
        x=1;
        state1=(PORTB&AND);     //霍尔信号采集
        DELAY1(x);
        state2=(PORTB&AND);
    }while(state1-state2);      //当三次采样结果不相同时继续采集状态
    if(state1-oldstate!=0)      //看本次采样结果是否与上次相同，不同
                                //则执行
     {oldstate=state1;          //将本次状态设置为旧状态
      state1=(oldstate>>5);
      PORTC=new[state1];            //C 口输出相应的信号触发两个 MOS 管
      if(sp1==1){spe=1;sp1=0;}
      else  {                   //如果转速很低，则 spe 置 1
```

```
        spe=0;sp1=0;
        speedcount<<=1;
        state3=(TMR1H>>2);      //否则，spe=0,计转速
        speed=speedcount+state3;    //speed 寄存器为每 256 μs 加 1
      }
      speedcount=0;
    }
}
//-----------------AD 采样子程序----------------------
void AD()
{
    char x;
    ADIF=0;                 //清 AD 中断标志位
    if(ts==1){              //如果为手柄采样，则采样手柄值
     CHS0=1;                    //选择电流采样通道
     count_vol=count_vol+1; //电池采样计数寄存器
     spepid=1;              //置转速闭环运算标志
     ts=0;tsh=ADRESH;       //存手柄值
     if(count_vol==0) { //如果电池采样时间到，则选择 AN2 通道，采集电池电压
        CHS0=0;CHS1=1;volflag=1;x=1;DELAY1(x);ADGO=1;
      }
    }
    else if(volflag==1) {           //电池采样完毕，进行相应的处理
       CHS1=0;CHS0=1;volflag=0;voltage=ADRESH;lowpower=1;
    }
    else    {                       //否则，中断为采样电流中断
      speedcount=speedcount+1;  //speedcount 寄存器加 1，作为测量转速用
      if(speedcount>0x3d)  sp1=1;   //如果转速低于 1 000 000 μs/(512 μs*3eh*3)
                                  // 则认为为低速状态
      currenth=ADRESH;
      curpid=1;
      count_ts=count_ts-1;
      if(count_ts==0) {         //如果手柄时间到，则转入手柄采样通道
          CHS0=0;count_ts=0x08;ts=1;x=1;DELAY1(x);ADGO=1;
        }
     }
}
//-------------刹车处理子程序------------------
void BREAKON()
{
    char x;
    off=0;                  //off 清零，如果是干扰则不复位
    shutdown=0;
    if(RB0==1)  {           //如果刹车信号为真，则停止输出电压
        ADIE=0;             //关 AD 中断
        INTE=0;             //关刹车中断
        CCPR1L=FULLDUTY;        //输出电压 0
        TMR1ON=0;           //关 CCP2，不再触发 AD
        for(;ADGO==1;)  continue;//如正在采样，则等待采样结束
        ADIF=0;             //ADIF 位清零
        CHS0=0;             //选择通道 0 采样手柄
```

```
        CHS1=0;
        x=1;
        DELAY1(x);
        do  {
            ADGO=1;
            for(;ADIF==0;)continue;
            ADIF=0;
            CCPR1L=FULLDUTY;
        asm("CLRWDT");
            tsh=(ADRESH>>1);
        }while(tsh>TSON||RB0==1);   //当手柄值大于 2.2 V 或刹车仍旧继续时,执行以
                                     //上语句
        off=1;                      //置复位标志
  }
}
//---------欠保护子程序-------------------
void POWER()
{
    char x;
    lowpower=0;
    voltage>>=1;                        //电压值换为 7 位，以利于单字节运算
    if(voltage<VOLOFF) {                //电池电压小于 3*k(V)时保护
        ADIE=0;
        INTE=0;
        TMR1ON=0;
        CCPR1L=FULLDUTY;
        for(;ADGO==1;)continue;
        ADIF=0;
        CHS0=0;CHS1=1;
        x=1;
        DELAY1(x);
        do{ADGO=1;
            for(;ADIF==0;)continue;
            ADIF=0;
            voltage=(ADRESH>>1);
            CCPR1L=FULLDUTY;
        asm("CLRWDT");
        }while(voltage<VOLON);  //电池电压小于 35 V 时继续保护
        off=1;                  //置复位标志
    }
}
//------------电流环运算子程序-----------------
void CURPI()
{   static int curep=0x00,curek=0x00,curuk=0x00;
    union data{int pwm;
        char a[2];}b;           //定义电流环运算寄存器
    curpid=0;                   //清电流运算标志
    curep=curek*CURB;           //计算上一次偏差与比例系数的积
    if(currenth<2)currenth=2;       //如果采样电流为零,则认为有一个小电流以利于
                                 //使转速下降
    currenth>>=1;
```

```
    curek=gcur-currenth;            //计算本次偏差
    curuk=curuk+curek*CURA-curep;       //按闭环 PI 运算方式得到本次输出结果，下
                                    //面对结果进行处理
    if(curuk<0x00) {                //如果输出小于零，则认为输出为零
      curuk=0;CCPR1L=FULLDUTY;CCP1X=0;CCP1Y=0;
      }
        else if(curuk-THL>=0)   {   //如果输出大于限幅值，则输出最大电压
            curuk=THL;CCPR1L=0;CCP1X=0;CCP1Y=0;
        }
        else    {               //否则,按比例输出相应的高电平时间到 CCPR1 寄存器
            b.pwm=THL-curuk;
            b.pwm<<=1;
            CCPR1L=b.a[1];  //CCPR1L=(b.pwm>>8)&0x0ff;将 PWM 寄存器的高半字节
            if(b.pwm&0x80!=0) CCP1X=1;
            else CCP1X=0;
            if(b.pwm&0x40!=0) CCP1Y=1;
            else CCP1Y=0;
    }
}
//---------------转速环运算子程序----------------------
void SPEPI()
{   static int speep=0x00,speek=0x00,speuk=0x00;
    int tsh1,speed1;                //转速寄存器定义
    spepid=0;                   //清转速运算标志
    if(spe==1)  speed1=0x00;        //若转速太低，则认为转速为零
    else speed1=0x7f-speed;     //否则计算实际转速
    if(speed1<0) speed1=0;
    speep=speek*SPEB;
    tsh1=tsh-0x38;              //得到计算用的手柄值
    speek=tsh1-speed1;
    if(tsh1<0) {speuk=0;gcur=0;}    //当手柄值低于 1.1 V 时，则认为手柄给定为零
    else {                      //否则，计算相应的转速环输出
        if(tsh1>=GSPEH)         //限制最大转速
        tsh1=GSPEH;
        speuk=speuk+speek*SPEA-speep;   //计算得转速环输出
        if(speuk<=0X00) {speuk=0x00;gcur=0x00;}//转速环输出处理
        else if(speuk>GCURHILO) {   //转速环输出限制，即限制最大电流约 12 A
            speuk=GCURHILO;gcur=GCURH;}
        else        {               //调速状态时的输出
            gcur=(speuk>>4)&0x0ff;
        }
    }
}
//-----------主程序------------------------
main()
{
    for(;;){
        INIT877();              //单片机复位后，先对其进行初始化
        off=0;                  //清复位标志
        for(;off==0;)   {       //复位标志为零，则执行下面程序，否则复位
        if(curpid==1) CURPI();  //电流 PI 运算
```

```
        else if(spepid==1)  SPEPI();    //转速 PI 运算
        else if(lowpower==1)    POWER();
        else if(shutdown==1)    BREAKON();
        asm("CLRWDT");
    }
  }
}
//---------中断服务子程序---------------------
#pragma interrupt_level 1
void interrupt INTS(void)
{
    if(RBIF==1) {RBIF=0;sample();}
    else if(ADIF==1)    AD();
    else if(INTF==1)    {shutdown=1;INTF=0;}    //刹车中断来，置刹车标志
}
```

NOTE：

第 10章　液晶显示模块编程

10.1 MG-12232 模块的编程

下面以图 7.1 的接口电路为例。液晶显示区域分成 E1 边和 E2 边，下面只含 E1 边的程序（表 7.1 中 E1=1，E2=0)，E2 边（表 7.1 中 E1=0，E2=1）类推。

在系统程序的初始化部分，应对程序中用到的寄存器和临时变量作说明，如：

```
unsigned char      TRANS；
unsigned char      PAGEADD； //存放页地址寄存器
unsigned char      PAGENUM； //存放总页数寄存器
unsigned char      CLMSUM；  //存放总列数寄存器
unsigned char      CLMADD；  //存放列地址寄存器
unsigned char      WRITE；   //存放显示数据寄存器
unsigned char      row；     //存放显示起始行寄存器
unsigned char      i，k；    //通用寄存器
//系统各口的输入输出状态初始化子程序
void INITIAL()
{
    ADCON1=0X87；                //设置 PORTA 口和 PORTE 口为数字 I/O 口
    TRISA3=0；
    TRISB0=0；
    TRISE=0X00；                 //设置液晶的 4 个控制脚为输出
}
//读液晶显示器状态子程序
void LCDSTA1()
{
    while(1)   {
        TRISD=0XFF；             //设置 D 口为输入
        RB0=1；                  //E1=1
        RA3=0；                  //E2=0
        RE0=1；                  //R/W=1
        RE1=0；                  //A0=0
        if(RD7==0)   break；     //为忙状态，则继续等待其为空闲
  }
}
//对液晶显示器发指令子程序（指令保存在 TRANS 寄存器中)
void TRANS1()
{
    LCDSTA1()；              //判断液晶是否为忙
    TRISD=0X00；             //置 D 口为输出
    RB0=1；                  //E1=1
    RA3=0；                  //E2=0
    RE0=0；                  //R/W=0
    RE1=0；                  //A0=0
    PORTD=TRANS；                //需要写入的命令字送入数据线
    RB0=0；                  //E1=0 写入指令
    RE0=1；                  //R/W=1
}
//对液晶显示器写数据子程序（数据保存在 WRITE 寄存器中）
void WRITE1()
{
    TRANS=CLMADD；           //设置列地址
```

```
        TRANS1()；
        LCDSTA1()；                  //查询液晶是否为空闲
        TRISD=0X00；                 //D 口为输出
        RB0=1；//E1=1
        RA3=0；//E2=0
        RE0=0；//R/W=0
        RE1=1；//A0=1
        PORTD=WRITE；                //需要写入的数据放入 D 口
        RB0=0；                      //E1=0，写入数据
        CLMADD++；                   //列地址加 1
        RE0=1；                      //R/W=1
}
//开 E1 显示子程序
void DISP1()
{
while(1)
    {
        TRANS=0XAF；
        TRANS1()；                   //送出控制命令
        LCDSTA1()；                  //判断液晶是否为空闲
        TRISD=0XFF；                 //设置 D 口为输入
        RB0=1；                      //E1=1
        RA3=0；                      //E2=0
        RE0=1；                      //R/W=1
        RE1=0；                      //A0=0
        if(RD5==0)    break；        //如果液晶没被关闭，则继续关
        }
}
//E1 边清屏子程序
void CLEAR1()
{
        PAGEADD=0xB8；               //设置页地址代码
        for(PAGENUM=0X04；PAGENUM>0；PAGENUM--){
                TRANS=PAGEADD；
                TRANS1()；
                CLMADD=0x00；                //设置起始列
                for(CLMSUM=0X50；CLMSUM>0；CLMSUM--){
                        LCDSTA1()；          //判断液晶是否为空闲
                        WRITE=0X00；
                        WRITE1()；           //写入 00H 以清屏
                }
                PAGEADD++；                  //页号增 1
          }
}
//关  E1 显示子程序
void        DISOFF1()
{
    while(1)
    {
        TRANS=0XAE；
        TRANS1()；                   //发出控制命令
        LCDSTA1()；                  //判断液晶是否为空闲
        TRISD=0XFF；                 //D 口设置为输入
```

```
      RB0=1；                    //E1=1
      RA3=0；                    //E2=0
      RE0=1；                    //R/W=1
      RE1=0；                    //A0=0
      if(RD5==1)    break；      //如果液晶没被关闭，则继续关
   }
}
```

有了以上的通用子程序，就可以构造出各种显示程序，如字符、汉字、曲线等。执行这些程序前，必须对液晶进行初始化。初始化的顺序为：关显示→正常显示驱动设置→占空比设置→复位→ADC 选择→清屏→开显示，程序如下：

```
//E1 边初始化
void      lcd1()
{
       DISOFF1()；                    //关显示 E1
       TRANS=0XA4；              //静态显示驱动
       TRANS1()；                //发出控制命令
       TRANS=0XA9；              //占空比为 1/32
       TRANS1()；                //发出控制命令
       TRANS=0XE2；              //复位
       TRANS1()；                //发出控制命令
            TRANS=0XA0；              //ADC 选择正常输出
            TRANS1()；                    //发出控制命令
       CLEAR1()；                //清屏
       LCDSTA1()；               //判断液晶是否为空闲
       DISP1()；                 //开显示
}
```

10.2　程序清单

下面给出一个已经在模板上调试通过的程序。　　注意在调试该程序时,需把模板上的 J9 跳针短接。

```
#include       <pic.h>
//该程序用于液晶显示功能的演示
//运行程序后，液晶上显示"电流有效值 "和"电压有效值 "字样
//系统总的初始化子程序
unsigned      char      TRANS；
unsigned      char      PAGEADD； //存放页地址寄存器
unsigned      char      PAGENUM；//存放总页数寄存器
unsigned      char      CLMSUM； //存放总列数寄存器
unsigned      char      CLMADD； //存放列地址寄存器
unsigned      char      WRITE；      //存放显示数据寄存器
unsigned      char      row；           //存放显示起始行寄存器
unsigned      char      i，k；          //通用寄存器
const   char   table[192]={0X00，0XF8，0X48，0X48，0X48，0X48，0XFF，0X48，
     0X48，0X48，0X48，0XFC，0X08，0X00，0X00，0X00，
     0X00，0X07，0X02，0X02，0X02，0X02，0X3F，0X42，
     0X42，0X42，0X42，0X47，0X40，0X70，0X00，0X00，//"电"
     0X00，0X00，0XFE，0X02，0X82，0X82，0X82，0X82，
     0XFE，0X82，0X82，0X82，0XC3，0X82，0X00，0X00，
     0X40，0X30，0X0F，0X40，0X40，0X40，0X40，0X40，
     0X7F，0X40，0X42，0X44，0X4C，0X60，0X40，0X00，//"压"
```

```
    0X04，0X04，0X04，0X84，0XE4，0X3C，0X27，0X24，
    0X24，0X24，0X24，0XF4，0X24，0X06，0X04 ，0X00，
    0X4 ，0X2 ，0X1 ，0X0 ，0XFF，0X9，0X9 ，0X9，
    0X9 ，0X49，0X89，0X7F，0X0，0X0，0X0 ，0X0，      //"有"
    0X88，0X48，0XB8，0X9，0XA，0X98，0X2C ，0X48，
    0X20，0XD0，0X1F，0X10，0X10，0XF8，0X10 ，0X0，
    0X40，0X20，0X18，0X5，0X2，0XD，0X30 ，0X80，
    0X80，0X41，0X36，0X8，0X37，0XC0，0X40 ，0X0，   //"效"
    0X80，0X40，0X20，0XF8，0X7，0X4，0XE4，0XA4，
    0XA4，0XBF，0XA4，0XA4，0XF6，0X24，0X0 ，0X0，
    0X0，0X0，0X0，0XFF，0X40，0X40，0X7F，0X4A，
    0X4A，0X4A，0X4A，0X4A，0X7F，0X40，0X40 ，0X0，//"值"
    0X10，0X22，0X64，0XC，0X80，0X44，0X44，0X64，
    0X55，0X4E，0X44，0X54，0X66，0XC4，0X0，0X0，
    0X4，0X4，0XFE，0X1，0X0，0X80，0X40，0X3F，
    0X0，0XFF，0X0，0X3F，0X40，0X40，0X70，0X0       //"流"
}；
//系统各口的输入输出状态初始化子程序
void      INITIAL()
{
    ADCON1=0X87；              //设置 PORTA 口和 PORTE 口为数字 I/O 口
    TRISA3=0；
    TRISB0=0；
    TRISE=0X00；               //设置液晶的 4 个控制脚为输出
}
//读液晶显示器状态子程序
void      LCDSTA1()
{
  while(1){
        TRISD=0XFF；              //设置 D 口为输入
        RB0=1；                   //E1=1
    RA3=0；                    //E2=0
    RE0=1；                    //R/W=1
    RE1=0；                    //A0=0
    if(RD7==0)   break；       //为忙状态，则继续等待其为空闲
   }
}
//对液晶显示器发指令子程序（指令保存在 TRANS 寄存器中)
void      TRANS1()
{
    LCDSTA1()；           //判断液晶是否为忙
    TRISD=0X00；          //D 口为输出
    RB0=1；               //E1=1
    RA3=0；               //E2=0
    RE0=0；               //R/W=0
    RE1=0；               //A0=0
    PORTD=TRANS；             //需要写入的命令字送入数据线
    RB0=0；               //E1=0 写入指令
    RE0=1；               //R/W=1
}
//对液晶显示器写数据子程序（数据保存在 WRITE 寄存器中）
void      WRITE1()
```

```
{
        TRANS=CLMADD；  //设置列地址
        TRANS1()；
        LCDSTA1()；              //查询液晶是否为空闲
        TRISD=0X00；             //D 口为输出
        RB0=1；                  //E1=1
        RA3=0；                  //E2=0
        RE0=0；                  //R/W=0
        RE1=1；                  //A0=1
        PORTD=WRITE；            //需要写入的数据放入 D 口
        RB0=0；                  //E1=0，写入数据
        CLMADD++；               //列地址加 1
        RE0=1；                  //R/W=1
}
//开 E1 显示子程序
void        DISP1()
{
        while(1)    {
        TRANS=0XAF；
        TRANS1()；                      //送出控制命令
        LCDSTA1()；                     //判断液晶是否为空闲
        TRISD=0XFF；                    //设置 D 口为输入
        RB0=1；                         //E1=1
        RA3=0；                         //E2=0
        RE0=1；                         //R/W=1
        RE1=0；                         //A0=0
        if(RD5==0)     break；          //如果液晶没被关闭，则继续关
        }
}
//E1 边清屏子程序
void        CLEAR1()
{
        PAGEADD=0xB8；                  //设置页地址代码
        for(PAGENUM=0X04；PAGENUM>0；PAGENUM--){
        TRANS=PAGEADD；
        TRANS1()；
        CLMADD=0x00；                   //设置起始列
        for(CLMSUM=0X50；CLMSUM>0；CLMSUM--) {
                LCDSTA1()；             //判断液晶是否为空闲
                WRITE=0X00；
                WRITE1()；              //写入 00H 以清屏
        }
        PAGEADD++；                     //页号增 1
    }
}
//关  E1 显示子程序
void        DISOFF1()
{
    while(1) {
        TRANS=0XAE；
        TRANS1()；               //发出控制命令
        LCDSTA1()；              //判断液晶是否为空闲
```

```
        TRISD=0XFF；            //D 口设置为输入
        RB0=1；                 //E1=1
        RA3=0；                 //E2=0
        RE0=1；                 //R/W=1
        RE1=0；                 //A0=0
        if(RD5==1)   break；    //如果液晶没被关闭，则继续关
      }
}
//E1 边初始化
void     lcd1()
{
         DISOFF1()；                 //关显示 E1
         TRANS=0XA4；           //静态显示驱动
         TRANS1()；             //发出控制命令
         TRANS=0XA9；           //占空比为 1/32
         TRANS1()；             //发出控制命令
         TRANS=0XE2；           //复位
         TRANS1()；             //发出控制命令
         TRANS=0XA0；           //ADC 选择正常输出
         TRANS1()；             //发出控制命令
         CLEAR1()；             //清屏
         LCDSTA1()；            //判断液晶是否为空闲
         DISP1()；              //开显示
}
//E2 边的处理部分
//读液晶显示器状态子程序
void     LCDSTA2()
{
        while(1) {
        TRISD=0XFF；            //设置 D 口为输入
        RB0=0；                 //E1=0
        RA3=1；                 //E2=1
        RE0=1；                 //R/W=1
        RE1=0；                 //A0=0
        if(RD7==0)   break；    //为忙状态，则继续等待其为空闲
    }
}
//对液晶显示器发指令子程序指令保存在 TRANS 寄存器中
void     TRANS2()
{
        LCDSTA2()；             //判断液晶是否为忙
        TRISD=0X00；            //D 口为输出
        RB0=0；                 //E1=0
        RA3=1；                 //E2=1
        RE0=0；                 //R/W=0
        RE1=0；                 //A0=0
        PORTD=TRANS；                //需要写入的命令字送入数据线
        RA3=0；                 //E2=0 写入指令
        RE0=1；                 //R/W=1
}
//对液晶显示器写数据子程序（数据保存在 WRITE 寄存器中）
void     WRITE2()
```

```
{
    TRANS=CLMADD；    //设置列地址
    TRANS2()；
    LCDSTA2()；       //查询液晶是否为空闲
    TRISD=0X00；      //D 口为输出
    RB0=0；           //E1=0
    RA3=1；           //E2=1
    RE0=0；           //R/W=0
    RE1=1；           //A0=1
    PORTD=WRITE；     //需要写入的数据放入 D 口
    RA3=0；           //E2=0，写入数据
    CLMADD++；        //列地址加 1
    RE0=1；           //R/W=1
}
//开 E2 显示子程序
void    DISP2()
{
    while(1)  {
    TRANS=0XAF；
    TRANS2()；        //送出控制命令
    LCDSTA2()；       //判断液晶是否为空闲
    TRISD=0XFF；      //设置 D 口为输入
    RB0=0；           //E1=0
    RA3=1；           //E2=1
    RE0=1；           //R/W=1
    RE1=0；           //A0=0
    if(RD5==0)   break；  //如果液晶没被关闭，则继续关
  }
}
//E2 边清屏子程序
void    CLEAR2()
{
    PAGEADD=0xB8；    //设置页地址代码
    for(PAGENUM=0X04；PAGENUM>0；PAGENUM--) {
        TRANS=PAGEADD；
        TRANS2()；
        CLMADD=0x00；     //设置起始列
        for(CLMSUM=0X50；CLMSUM>0；CLMSUM--) {
            LCDSTA2()；   //判断液晶是否为空闲
            WRITE=0X00；
            WRITE2()；    //写入 00H 以清屏
        }
        PAGEADD++；       //页号增 1
  }
}
//关  E2 显示子程序
void    DISOFF2()
{
    while(1) {
    TRANS=0XAE；
    TRANS2()；        //发出控制命令
    LCDSTA2()；       //判断液晶是否为空闲
```

```
        TRISD=0XFF；            //D 口设置为输入
        RB0=0；                 //E1=0
        RA3=1；                 //E2=1
        RE0=1；                 //R/W=1
        RE1=0；                 //A0=0
        if(RD5==1)   break；    //如果液晶没被关闭，则继续关
      }
}
//E2 边初始化
void      lcd2()
{
        DISOFF2()；             //关显示 E1
        TRANS=0XA4；            //静态显示驱动
        TRANS2()；              //发出控制命令
        TRANS=0XA9；            //占空比为 1/32
        TRANS2()；              //发出控制命令
        TRANS=0XE2；            //复位
        TRANS2()；              //发出控制命令
        TRANS=0XA0；            //ADC 选择正常输出
        TRANS2()；              //发出控制命令
        CLEAR2()；              //清屏
        LCDSTA2()；             //判断液晶是否为空闲
        DISP2()；               //开显示
}
//LCD 的 E1 边显示函数，调用一次该函数，则在相应的位置显示相应的字
void      dis1()
{
         TRANS=row；
         TRANS1()；
         TRANS=PAGEADD；
         TRANS1()；
         i=i*32；                          //i 变成数组指示指针
         for(k=0；k<16；k++)  {
              WRITE=table[i+k]；   //查得需要显示的字节
              WRITE1()；           //在 WRITE1 子程序里面，列地址加 1
         }
         CLMADD=CLMADD-16；//恢复列地址
         PAGEADD=PAGEADD+1；//页地址加 1
         TRANS=PAGEADD；
         TRANS1()；
         for(；k<32；k++)      {
              WRITE=table[i+k]；   //查得需要显示的字节
              WRITE1()；           //在 WRITE1 子程序里面，列地址已经加 1
         }
}
//LCD 的 E2 边显示函数，调用一次该函数，则在相应的位置显示相应的字
void      dis2()
{
         TRANS=row；
         TRANS2()；
         TRANS=PAGEADD；
         TRANS2()；
```

```
        i=i*32；                          //i 变成数组指示指针
        for(k=0；k<16；k++)   {
              WRITE=table[i+k]；   //查得需要显示的字节
              WRITE2()；           //在 WRITE1 子程序里面，列地址已经加 1
        }
        CLMADD=CLMADD-16；//恢复列地址
        PAGEADD=PAGEADD+1；//页地址加 1
        TRANS=PAGEADD；
        TRANS2()；
        for(；k<32；k++)        {
            WRITE=table[i+k]；    //查得需要显示的字节
              WRITE2()；           //在 WRITE1 子程序里面，列地址已经加 1
        }
}
//主程序
main()
{
      INITIAL()；                  //系统初始化
      lcd1()；                     //E1 边初始化
      lcd2()；                     //E2 边初始化
      row=0XC0；                   //显示起始列为第 0 行
//以下显示不同的字符
      PAGEADD=0XB8；               //显示起始页为第 0 页
      CLMADD=0X00；                //起始列为第 0 列
      i=0；                        //显示数组中对应的第一个字
      dis1()；                     //调用显示函数
      PAGEADD=0XB8；               //显示起始页为第 0 页
      CLMADD=16；                  //起始列为第 16 列
      i=1；                        //显示数组中对应的第二个字
      dis1()；                     //调用显示函数
      PAGEADD=0XB8；               //显示起始页为第 0 页
      CLMADD=32；                  //起始列为第 32 列
      i=2；                        //显示数组中对应的第三个字
      dis1()；                     //调用显示函数
      PAGEADD=0XB8；               //显示起始页为第 0 页
      CLMADD=48；                  //起始列为第 48 列
      i=3；                        //显示数组中对应的第四个字
      dis1()；                     //调用显示函数
      PAGEADD=0XB8；               //显示起始页为第 0 页
      CLMADD=0；                   //起始列为第 0 列
      i=4；                        //显示数组中对应的第五个字
      dis2()；                     //调用 E2 边显示函数
      PAGEADD=0XBA；               //显示起始页为第 2 页
      CLMADD=0X00；                //起始列为第 0 列
      i=0；                        //显示数组中对应的第一个字
      dis1()；                     //调用显示函数
      PAGEADD=0XBA；               //显示起始页为第 2 页
      CLMADD=16；                  //起始列为第 16 列
      i=5；                        //显示数组中对应的第六个字
      dis1()；                     //调用显示函数
      PAGEADD=0XBA；               //显示起始页为第 2 页
```

```
    CLMADD=32；              //起始列为第 32 列
    i=2；                    //显示数组中对应的第三个字
    dis1()；                 //调用显示函数
    PAGEADD=0XBA；           //显示起始页为第 2 页
    CLMADD=48；              //起始列为第 48 列
    i=3；                    //显示数组中对应的第四个字
    dis1()；                 //调用显示函数
    PAGEADD=0XBA；           //显示起始页为第 2 页
    CLMADD=0；               //起始列为第 0 列
    i=4；                    //显示数组中对应的第五个字
    dis2()；                 //调用 E2 边显示函数
    while(1)  {
        ；
    }
}
```

NOTE：